Refine Search

Search Results -

Terms	Documents
L2 and (patch$3.ti. or patch$3.ab.)	25

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L3

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, March 02, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L3	L2 and (patch$3.ti. or patch$3.ab.)	25	L3
L2	l1 and download$3	970	L2
L1	Patch$3 and database?	3427	L1

Scanned all

END OF SEARCH HISTORY

Print Request Result(s)

Printer Name: cpk2_5d57_gbkjptr
Printer Location: cpk2__5d57
Number of Copies Printed : 1

- US20020174422: Ok
- US20040015938: Ok
- US20040003266: Ok
- US006438749: Ok
- US20040015949: Ok
- US20020112230: Ok
- US20020116665: Ok

OK Back to List

First Hit

Generate Collection Print

L3: Entry 2 of 25 File: PGPB Jan 22, 2004

PGPUB-DOCUMENT-NUMBER: 20040015949
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040015949 A1

TITLE: Method, system, program, and data structures for applying a patch to a computer system

PUBLICATION-DATE: January 22, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Taylor, Julian S.	Nederland	CO	US	

US-CL-CURRENT: 717/171; 717/121, 717/168

CLAIMS:

What is claimed:

1. A method for applying a patch to a computer system, wherein the patch includes content to add to the computer, comprising: determining at least one of installed hardware and software components on the computer; generating a computer object including configuration information on the determined installed components; providing at least one patch including content to add to the computer, wherein each patch is capable of being associated with at least one realization, wherein each realization defines a state of the computer; for each realization, determining from the configuration information in the computer object whether the state defined by the realization exists in the computer; writing data to the computer object indicating whether the state defined by the realization exists on the computer; and using the computer object to determine whether each patch is compatible with the installed components of the computer.

2. The method of claim 1, wherein the patch content to add to the computer is capable of comprising one of: a new program; an upgrade to an installed program; a fix to an installed program; and documentation.

3. The method of claim 1, wherein one realization is capable of being dependent on another realization in the computer object, further comprising: determining whether the realization is dependent on at least one base realization; and if the realization is dependent on at least one base realization, performing: (i) determining whether the computer object includes information on the base realizations; and (ii) if the computer object does not include information on the base realizations, then throwing an exception.

4. The method of claim 3, further comprising: in response to throwing the exception, preventing each patch associated with at least one realization dependent from the base realization not written to the computer object from being installed.

5. The method of claim 1, wherein writing data to the computer object indicating that the state exists comprises: indicating in the computer object that the state defined by the realization exists in the computer if the state exists on the computer; and indicating in the computer object that the state defined by the realization does not exist on the computer if the state does not exist on the computer.

6. The method of claim 1, wherein determining from the computer object whether the state exists on the computer includes determining whether information on one previously considered realization is written to the computer object.

7. The method of claim 1, wherein each patch is further associated with a patch expression that is capable of processing the computer object to determine whether to add the content from the patch onto the computer.

8. The method of claim 1, further comprising downloading one or more realization detectors, wherein each realization detector is used to check at least one realization, and wherein each realization detector includes detector code to determine whether states defined by realizations exist on the computer and to write data to the computer object concerning the existence of the determined states.

9. The method of claim 1, further comprising: downloading a plurality of patches to the computer, each including content to add to the computer, wherein the steps of executing the realization routine, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer is performed on the computer.

10. The method of claim 9, further comprising: generating a list of all patches determined to be compatible with the installed components of the computer; and rendering the list of patches on an output device to enable a user to select the content of one or more of the patches on the list to add to the computer.

11. The method of claim 1, further comprising: maintaining, at a network administrator system, a plurality of computer objects associated with a plurality of computers on a network, wherein each computer object includes the configuration information on the determined installed components of one computer on the network and wherein the steps of determining whether the state defined by the realization exists on the computer, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer on the network is performed for computer objects on the network administrator system.

12. The method of claim 11, further comprising: maintaining a plurality of patches at the network administrator system, wherein determining the states defined by the realizations, writing data to the computer object, and using the computer objects to determine whether the patch is compatible with the installed components of the computers on the network is performed for each patch available to the network administrator system.

13. The method of claim 12, further comprising: generating a list of all patches determined to be compatible with the installed components of the computers on the network; and rendering the list of patches on an output device to enable a user of the network administrator system to select the code of one or more of the patches on the list to install on one or more of the computers in the network.

14. The method of claim 1, wherein the content comprises a fix, and wherein the state defined by the realization is capable of indicating whether the computer is susceptible to a bug corrected by the fix.

15. The method of claim 1, wherein determining the at least one of installed hardware and software components comprises: executing a detector program on the computer to process files in the computer to determined installed hardware and software components on the computer.

16. The method of claim 15, wherein determining the at least one of installed hardware and software components comprises: receiving user input indicating installed hardware and software components on the computer.

17. The method of claim 16, wherein the user input is received by: displaying a user interface presenting user selectable software and hardware components; and receiving user selection of software and hardware components from the displayed user interface, wherein the received user input indicating the installed hardware and software components comprises the components selected from the user interface.

18. The method of claim 1, wherein reading and writing performed as part of adding content to the computer is restricted to the computer object on the computer.

19. A system for applying a patch, wherein the patch includes content to add, comprising: means for determining at least one of installed hardware and software components on the computer; means for generating a computer object including configuration information on the determined installed components; means for providing at least one patch including content to add to the computer, wherein each patch is capable of being associated with at least one realization, wherein each realization defines a state of the computer; means for determining from the configuration information in the computer object for each realization whether the state defined by the realization exists in the computer; means for writing data to the computer object indicating whether the state defined by the realization exists on the computer; and means for using the computer object to determine whether each patch is compatible with the installed components of the computer.

20. The system of claim 19, wherein the patch content to add to the computer is capable of comprising one of: a new program; an upgrade to an installed program; a fix to an installed program; and documentation.

21. The system of claim 19, wherein one realization is capable of being dependent on another realization in the computer object, further comprising: means for determining whether the realization is dependent on at least one base realization; and means for performing, if the realization is dependent on at least one base realization: (i) determining whether the computer object includes information on the base realizations; and (ii) if the computer object does not include information on the base realizations, then throwing an exception.

22. The system of claim 21, further comprising: means for preventing each patch associated with at least one realization dependent from the base realization not written to the computer object from being installed in response to throwing the exception.

23. The system of claim 19, wherein the means for writing data to the computer object indicating that the state exists further performs: indicating in the computer object that the state defined by the realization exists in the computer if the state exists on the computer; and indicating in the computer object that the state defined by the realization does not exist on the computer if the state does not exist on the computer.

24. The system of claim 19, wherein the means for determining from the computer object whether the state exists on the computer further performs determining whether information on one previously considered realization is written to the

computer object.

25. The system of claim 19, wherein each patch is further associated with a patch expression, further comprising means for processing the patch and the computer object to determine whether to add the content from the patch onto the computer.

26. The system of claim 19, further comprising: means for downloading one or more realization detectors, wherein each realization detector is used to check at least one realization, and wherein each realization detector includes detector code to determine whether states defined by realizations exist on the computer and to write data to the computer object concerning the existence of the determined states.

27. The system of claim 19, further comprising: means for downloading a plurality of patches to the computer, each including content to add to the computer, wherein the means for executing the realization routine, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer is performed in the computer.

28. The system of claim 27, further comprising: means for generating a list of all patches determined to be compatible with the installed components of the computer; and means for rendering the list of patches on an output device to enable a user to select the content of one or more of the patches on the list to add to the computer.

29. The system of claim 19, further comprising: means for maintaining a plurality of computer objects associated with a plurality of computers on a network, wherein each computer object includes the configuration information on the determined installed components of one computer on the network.

30. The system of claim 29, further comprising: means for maintaining a plurality of patches, wherein determining the states defined by the realizations, writing data to the computer object, and using the computer objects to determine whether the patch is compatible with the installed components of the computers on the network is performed for each patch available to the network administrator system.

31. The system of claim 32, further comprising: means for generating a list of all patches determined to be compatible with the installed components of the computers on the network; and means for rendering the list of patches on an output device to enable a user of the network administrator system to select the code of one or more of the patches on the list to install on one or more of the computers in the network.

32. The system of claim 19, wherein the content comprises a fix, and wherein the state defined by the realization is capable of indicating whether the computer is susceptible to a bug corrected by the fix.

33. The system of claim 19, wherein the means for determining the at least one of installed hardware and software components performs: executing a detector program on the computer to process files in the computer to determined installed hardware and software components on the computer.

34. The system of claim 33, wherein the means for determining the at least one of installed hardware and software components performs: receiving user input indicating installed hardware and software components on the computer.

35. The system of claim 34, further comprising: means for displaying a user interface presenting user selectable software and hardware components; and means

for receiving user selection of software and hardware components from the displayed user interface, wherein the received user input indicating the installed hardware and software components comprises the components selected from the user interface.

36. The system of claim 19, wherein the means for reading and writing performed as part of adding content to the computer is restricted to the computer object on the computer.

37. An article of manufacture for applying a patch to a computer system, wherein the patch includes content to add to the computer, by: determining at least one of installed hardware and software components on the computer; generating a computer object including configuration information on the determined installed components; providing at least one patch including content to add to the computer, wherein each patch is capable of being associated with at least one realization, wherein each realization defines a state of the computer; for each realization, determining from the configuration information in the computer object whether the state defined by the realization exists in the computer; writing data to the computer object indicating whether the state defined by the realization exists on the computer; and using the computer object to determine whether each patch is compatible with the installed components of the computer.

38. The article of manufacture of claim 37, wherein the patch content to add to the computer is capable of comprising one of: a new program; an upgrade to an installed program; a fix to an installed program and documentation.

39. The article of manufacture of claim 37, wherein one realization is capable of being dependent on another realization in the computer object, further comprising: determining whether the realization is dependent on at least one base realization; and if the realization is dependent on at least one base realization, performing: (i) determining whether the computer object includes information on the base realizations; and (ii) if the computer object does not include information on the base realizations, then throwing an exception.

40. The article of manufacture of claim 39, further comprising: in response to throwing the exception, preventing each patch associated with at least one realization dependent from the base realization not written to the computer object from being installed.

41. The article of manufacture of claim 37, wherein writing data to the computer object indicating that the state exists comprises: indicating in the computer object that the state defined by the realization exists in the computer if the state exists on the computer; and indicating in the computer object that the state defined by the realization does not exist on the computer if the state does not exist on the computer.

42. The article of manufacture of claim 37, wherein determining from the computer object whether the state exists on the computer includes determining whether information on one previously considered realization is written to the computer object.

43. The article of manufacture of claim 37, wherein each patch is further associated with a patch expression that is capable of processing the computer object to determine whether to add the content from the patch onto the computer.

44. The article of manufacture of claim 37, further comprising downloading one or more realization detectors, wherein each realization detector is used to check at least one realization, and wherein each realization detector includes detector code to determine whether states defined by realization exist on the computer and to

write data to the computer object concerning the existence of the determined states.

45. The article of manufacture of claim 37, further comprising: downloading a plurality of patches to the computer, each including content to add to the computer, wherein the steps of executing the realization routine, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer is performed on the computer.

46. The article of manufacture of claim 45, further comprising: generating a list of all patches determined to be compatible with the installed components of the computer; and rendering the list of patches on an output device to enable a user to select the content of one or more of the patches on the list to add to the computer.

47. The article of manufacture of claim 37, further comprising: maintaining, at a network administrator system, a plurality of computer objects associated with a plurality of computers on a network, wherein each computer object includes the configuration information on the determined installed components of one computer on the network, and wherein the steps of determining whether the state defined by the realization exists on the computer, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer on the network is performed for computer objects on the network administrator system.

48. The article of manufacture of claim 47, further comprising: maintaining a plurality of patches at the network administrator system, wherein determining the states defined by the realizations, writing data to the computer object, and using the computer objects to determine whether the patch is compatible with the installed components of the computers on the network is performed for each patch available to the network administrator system.

49. The article of manufacture of claim 48, further comprising: generating a list of all patches determined to be compatible with the installed components of the computers on the network; and rendering the list of patches on an output device to enable a user of the network administrator system to select the code of one or more of the patches on the list to install on one or more of the computers m the network.

50. The article of manufacture of claim 37, wherein the content comprises a fix, and wherein the state defined by the realization is capable of indicating whether the computer is susceptible to a bug corrected by the fix.

51. The article of manufacture of claim 37, wherein determining the at least one of installed hardware and software components comprises: executing a detector program on the computer to process files in the computer to determined installed hardware and software components on the computer.

52. The article of manufacture of claim 51, wherein determining the at least one of installed hardware and software components comprises: receiving user input indicating installed hardware and software components on the computer.

53. The article of manufacture of claim 52, wherein the user input is received by: displaying a user interface presenting user selectable software and hardware components; and receiving user selection of software and hardware components from the displayed user interface, wherein the received user input indicating the installed hardware and software components comprises the components selected from

the user interface.

54. The article of manufacture of claim 37, wherein reading and writing performed as part of adding content to the computer is restricted to the computer object on the computer.

55. A computer readable medium including data structures used for applying a patch to a computer system, wherein the patch includes content to add to the computer, comprising: a computer object including configuration information on installed hardware and software components in the computer; a realization object including an association of at least one patch including content to add to the computer and at least one realization, wherein each realization defines a state of the computer, wherein the configuration information in the computer object is used to determine whether the state defined by the realization exists in the computer, wherein data is written to the computer object indicating whether the state defined by the realization exists on the computer, and wherein the computer object is used to determine whether each patch is compatible with the installed components of the computer.

56. The computer readable medium of claim 55, further comprising: a patch expression set including code that when executed processes the computer object to determine whether to add the content from the patch onto the computer.

57. The computer readable medium of claim 55, further comprising: a realization detector object including detector code to determine whether states defined by at least one realization exists on the computer and to write data to the computer object concerning the existence of the determined states.

58. The computer readable medium of claim 55, further comprising: a plurality of computer objects associated with a plurality of computers on a network, wherein each computer object includes the configuration information on the determined installed components of one computer on the network, and wherein the determination of whether the state defined by the realization exists on the computer, writing data to the computer object, and using the computer object to determine whether the patch is compatible with the installed components of the computer on the network is performed for computer objects on the network administrator system.

First Hit

☐ Generate Collection | Print

L3: Entry 12 of 25 File: PGPB Aug 15, 2002

DOCUMENT-IDENTIFIER: US 20020112230 A1
TITLE: Software update management system with update chronology generator

Abstract Paragraph:
An update management system provides access to software updates (patches and upgrades) and to an update catalog server. The catalog server includes a chronology generator. When a request for an update family tree for a target update is received, the chronology generator accesses an index to find the database and record number for the target update. The record is retrieved, indicating updates superseded by the target update and updates superseding the target update. Records for the superseded and superseding updates are, in turn, retrieved. The process is iterated until there are no further superseding and no superseded updates indicated. The succession relations indicated in the retrieved records are arranged into a family tree for the target patch. The family tree can be used (along with dependency data) to help determine an update for a given workstation that optimizes performance and compatibility. Optionally, the family tree can be used to generate an update state list to assist in the optimization.

Summary of Invention Paragraph:
[0003] Almost inevitably, it becomes necessary or desirable to modify a program: 1) to correct a defect, 2) to address a compatibility issue with other software or hardware, 3) to improve performance, and/or 4) to add features. (If no new features are added, the update is called a "patch"; if new features are added, the update is called an "upgrade".) Instead of replacing an entire program, the program can be updated. Updating can involve actually changing the instructions in a monolithic program. However, partly to facilitate updates, programs are often configured as a group of files so that an update can simply involve a replacing a preexisting file with an update file.

Summary of Invention Paragraph:
[0007] The present invention provides an update management system including an update chronology generator. In response to a request via a network received at an input of the system, the update chronology generator generates a chronology regarding a given target update. The chronology is generated by accessing a database record for the target update, as well as database records for updates indicated in the target-update record as being succeeded by the target update. The update management system can access a set of one or more databases including an internal database or an external database or both. Preferably, the chronology extends back to the "base" program, and also extends forward to indicate updates that succeed the target update.

Detail Description Paragraph:
[0011] In accordance with the present invention, an update management system AP1 comprises an update catalog server 10 and update files 11. Update catalog server 10 includes a chronology generator 13, a dependency generator 15, an update index 17, and a local update database 19. The chronology generator and the dependency generator share a common interface 20 for accessing local update database 19, as well as external databases. Update management system AP1 is connected to a network 21 at a network port 22, on which two update databases 23 and 25 reside. A remote workstation 31 on a remote network 33 communicates with update management system

AP1 via an inter-network link 35.

Detail Description Paragraph:
[0012] The invention provides for use of one or more databases; each of which can be either local (part of update management system AP1) or remote. Normally, when there are plural databases, they are mutually exclusive as to target updates represented. Alternatively, a local database can be used as a cache to improve performance.

Detail Description Paragraph:
[0013] Typically, requests are made to update management system AP1 from a remote workstation, such as remote workstation 31, and are received by update management system AP1 at network port 22. The requests are made for updates to a "base" program, typically residing on the remote workstation. The base program can be an operating system, a utility program, an application program, etc. The updates can be patches--which fix problems or improve performance without adding features, or upgrades, which add features, or both.

Detail Description Paragraph:
[0015] Of particular interest herein, are "chronology" requests for an update chronology for a target update. For example, a chronology of patches leading up to a target patch from the initial release of the base program. Alternatively, the chronology can specify updates that supersede the target update. Most useful, is a chronology that indicates an entire succession from the base program, through the target update, to its most recent successor.

Detail Description Paragraph:
[0018] It is often the case that an update will in effect merge two or more of its predecessors. Hence, the chain of succession up to a target update can have a tree-type structure. Updates that have a common successor are aligned vertically in the report. For example, PHKL.sub.--6000 and PHKL.sub.--7000 are aligned, since they are both succeeded by PHKL.sub.--8000. Likewise, PHKL.sub.--6000 succeeds PHKL.sub.--3000, PHKL.sub.--4000 and PHKL.sub.--5000. The report also indicates that PHKL.sub.--3000 supersedes only PHKL.sub.--1000, and PHKL.sub.--5000 supersedes only PHKL.sub.--2000. All updates for which no predecessor is indicated are patches to the base program.

Detail Description Paragraph:
[0020] A method M1 of the invention is flow-charted in FIG. 2. An update chronology request is received by update management system AP1 at step S1. Update catalog server 10 handles this request. In particular, chronology generator 13 accesses update index 17 to determine the location of an update record for the target update at step S2. Update index 17 lists updates alphabetically and indicates a database and a record number for each update listed therein. The database can be local database 19, considered part of update catalog server 10, or it can be an external database, such as databases 23 and 25 at remote nodes on network 21.

Detail Description Paragraph:
[0021] At step S3, chronology generator 13 accesses the database and target record identified in index 17. For small systems with not too many updates, index lookup step S2 is not necessary. However, the larger the number of records and databases, the more time is saved using the index in step S2.

Detail Description Paragraph:
[0025] The family tree or the state table can be used in upgrading a system when the most recent updates cause problems. For example, assume workstation 31 had PHKL.sub.--1000 and PHKL.sub.--2000 installed at its last update. Also, assume a user for workstation 31 is advised to upgrade to PHKL.sub.--8000 for higher performance. The user performs a chronology request and discovers that PHKL.sub.--8000 has been superseded by PHKL.sub.--9000. The user downloads and installs

PHKL.sub.--9000 from update manager system API. In addition, the user can issue a dependency request handled by dependency generator 15 to ensure all updates required by PHKL.sub.--9000 are installed.

Detail Description Paragraph:
[0028] While the invention applies generally to updates, it has been implemented in the following patch-family-tree tool (pft) for patches. The following command-line syntax with switches can be used.

Detail Description Paragraph:
[0029] Usage: pft -o <os> [-p <platform>] [-s <servers>] [-v] [-l] [-r] [-e] [-c] [-a] [-y] [-z] [-i <number_of_spaces>] <patch_name_or.sub.--number>

Detail Description Paragraph:
[0036] -l=show recalled patches

Detail Description Paragraph:
[0037] -r=show released patches

Detail Description Paragraph:
[0038] -e=show patches with dependencies

Detail Description Paragraph:
[0039] -c=show critical patches

Detail Description Paragraph:
[0040] -a=show reposted patches

Detail Description Paragraph:
[0041] -y=show superseded patches ("older")

Detail Description Paragraph:
[0042] -z=show superseding patches ("newer") (default is to show both)

Detail Description Paragraph:
[0046] A patch family tree is generated for a single patch and it must be specified on the command line.

Detail Description Paragraph:
[0048] The report can include one-line descriptions of each patch, by specifying the -v option, for "verbose" output.

Detail Description Paragraph:
[0049] By default, the report will contain patches that are both older than (superseded by) and newer than (supersede) the specified patch. Only the older patches are shown when the -y option is used. Only the newer patches are shown when the -z option is used.

Detail Description Paragraph:
[0050] Various attributes of the listed patches can be displayed using several options. If the patch has the requested attribute, a flag will be included in the output listing.

Detail Description Paragraph:
[0051] This following example shows the use of the -v (verbose) option and the -i (indentation) option, for the command line beginning with the "$"-sign below. The specified patch supersedes five patches: PHKL.sub.--14070, PHKL.sub.--14034, PHKL.sub.--13676, PHKL.sub.--13644, and PHKL.sub.--13328. PHKL.sub.--14070 supersedes one patch, PHKL.sub.--13858, which supersedes PHKL.sub.--13552, which supersedes PHKL.sub.--13081.

Detail Description Paragraph:
[0054] # Patch Family Tree

Detail Description Paragraph:
[0055] # PHKL.sub.--14088 HP-UX Performance Pack cumulative patch

Detail Description Paragraph:
[0058] # Only superseded patches are displayed.

Detail Description Paragraph:
[0059] # Superseded patches are indented to the right

Detail Description Paragraph:
[0061] # Patches aligned in a vertical column are in the

Detail Description Paragraph:
[0064] PHKL.sub.--14088 HP-UX Performance Pack; cumulative patch

Detail Description Paragraph:
[0065] PHKL.sub.--14070 Tape, IOCTL, FC fixes cumulative patch

Detail Description Paragraph:
[0066] .vertline. PHKL.sub.--13858 Tape and IOCTL fixes cumulative patch

Detail Description Paragraph:
[0069] PHKL.sub.--14034 SHMEM_MAGIC Perf, Mem window patch

Detail Description Paragraph:
[0078] The following example shows the use of the -y (display older patches) and the -r (denote recalled patch) options. Only patches that are superseded by the specified patch, PHCO.sub.--12922, are displayed. The patches that supersede PHCO.sub.--12992 (PHCO.sub.--12923,

Detail Description Paragraph:
[0079] PHCO.sub.--14842, and PHCO.sub.--16591) are not displayed. Note the two recalled patches, PHCO.sub.--11909 and PHCO.sub.--11908, flagged with the "RCL" keyword.

Detail Description Paragraph:
[0082] # Patch Family Tree

Detail Description Paragraph:
[0083] # PHCO.sub.--12922 fsck.sub.--vxfs(1M) cumulative patch

Detail Description Paragraph:
[0087] # Only superseded patches are displayed.

Detail Description Paragraph:
[0088] # Superseded patches are indented to the right

Detail Description Paragraph:
[0090] # Patches aligned in a vertical column are in the

Detail Description Paragraph:
[0092] # RCL=Recalled patch

Detail Description Paragraph:
[0094] PHCO.sub.--12922 fsck.sub.--vxfs(1M) cumulative patch

Detail Description Paragraph:
[0095] PHCO.sub.--11909 RCL fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0096] PHCO.sub.--11908 RCL fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0097] PHCO.sub.--11223 fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0098] PHCO.sub.--10965 fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0100] The recall notices for the "recalled" patches can be viewed using a query tool. The query tool, indicated by the abbreviation "qpc", is used to send a message to the catalog server. Quite often this message takes the form of a query. The query tool reads a message from its command line arguments, sends it to the specified server, waits for the answer, and displays.

Detail Description Paragraph:
[0103] Warn: 97/10/21--This patch has been recalled. --Patch PHCO.sub.--11909 can cause OmniStorage A.02.20 filesystems to be unmountable on HP-UX 10.20. The problem is . . .

Detail Description Paragraph:
[0104] The following example shows the use of the -z (display newer patches) option, and the -e (display patches with dependencies) option. Note that each of the patches that supersede PHCO.sub.--11909 has a dependency on at least one other patch.

Detail Description Paragraph:
[0107] # Patch Family Tree

Detail Description Paragraph:
[0108] # PHCO.sub.--11909 fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0112] # Only superseding patches are displayed.

Detail Description Paragraph:
[0113] # Superseding patches are indented to the left

Detail Description Paragraph:
[0115] # Patches aligned in a vertical column are in the

Detail Description Paragraph:
[0117] # DEP =Patch has dependencies on other patch(es)

Detail Description Paragraph:
[0118] # ODEP =Patch has other dependencies

Detail Description Paragraph:
[0120] PHCO.sub.--16591 DEP fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0121] PHCO.sub.--14842 DEP fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0122] PHCO.sub.--12923 DEP fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0123] PHCO.sub.--12922 DEP fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0124] PHCO.sub.--11909 DEP fsck_vxfs(1M) cumulative patch

Detail Description Paragraph:
[0125] The patch dependencies can be viewed using the patch dependency analysis tool "pdat", as implemented by patch dependency generator 15.

Detail Description Paragraph:
[0127] PHCO.sub.--18563 LVM commands cumulative patch

Detail Description Paragraph:
[0128] PHKL.sub.--16750 SIG_IGN/SIGCLD,LVM,JFS,PCI/SCSI cum. patch

Detail Description Paragraph:
[0129] PHKL.sub.--16959 Physical dump devices configuration patch

Detail Description Paragraph:
[0132] PHKL.sub.--21594 VxFS (JFS) mount, fsck cumulative patch

Detail Description Paragraph:
[0133] PHKL.sub.--21660 lo_realvfs panic fix, Cum. LOFS patch

Detail Description Paragraph:
[0134] PHNE.sub.--19937 cumulative ARPA Transport patch

Detail Description Paragraph:
[0135] The following example shows the use of the -c (flag critical patches) option for a patch family tree command. Patches which have been released with Support Plus and/or Extension Software are flagged when the -r option is specified. PHCO.sub.--14198 is a critical patch, and both PHCO.sub.--14198 and PHCO.sub.--13131 have been released in a Support Plus and/or Extension Software bundle.

Detail Description Paragraph:
[0138] # Patch Family Tree

Detail Description Paragraph:
[0139] # PHCO.sub.--14198 crashutil(1M) cumulative patch

Detail Description Paragraph:
[0142] # Both superseded and superseding patches are

Detail Description Paragraph:
[0144] # Superseded patches are indented to the right

Detail Description Paragraph:
[0146] # Superseding patches are indented to the left

Detail Description Paragraph:
[0148] # Patches aligned in a vertical column are in the

Detail Description Paragraph:
[0150] # REL=Patch released with Extension Software and/or

Detail Description Paragraph:
[0153] # CRIT=Patch contains critical fixes

Detail Description Paragraph:

[0154] PHCO.sub.--14198 REL CRIT crashutil(1M) cumulative patch

Detail Description Paragraph:
[0155] PHCO.sub.--13131 REL crashutil(1M) cumulative patch

Detail Description Paragraph:
[0156] Critical fix information for a patch, and all patches that it supersedes, can be viewed using qpctree:

Detail Description Paragraph:
[0164] The following example shows the use of the -a (flag reposted patches) option. In this example, PHCO.sub.--10576 has been reposted.

Detail Description Paragraph:
[0167] # Patch Family Tree

Detail Description Paragraph:
[0168] # PHCO.sub.--14967 sar(1M) cumulative patch

Detail Description Paragraph:
[0172] # Both superseded and superseding patches are

Detail Description Paragraph:
[0174] # Superseded patches are indented to the right

Detail Description Paragraph:
[0176] # Superseding patches are indented to the left

Detail Description Paragraph:
[0178] # Patches aligned in a vertical column are in the

Detail Description Paragraph:
[0180] # REP=Patch was reposted

Detail Description Paragraph:
[0181] PHCO.sub.--14967 sar(1M) cumulative patch

Detail Description Paragraph:
[0182] PHCO.sub.--14228 sar(1) cumulative patch

Detail Description Paragraph:
[0183] PHCO.sub.--10576 REP sar(1) cumulative patch

Detail Description Paragraph:
[0184] PHCO.sub.--8820 sar(1M) patch

Detail Description Paragraph:
[0189] A problem was discovered with replacement patch PHCO.sub.--14228. PHCO.sub.--14228 breaks the Year 2000 compliance implemented in patch PHCO.sub.--8820. PHCO.sub.--10576 will be re-released until a replacement patch is available.

Detail Description Paragraph:
[0190] As indicated earlier, the patch tools work as well for upgrades, so they are general to updates. The invention has industrial applicability to both hardware and software manufacturers, as it allows them to organize and distribute their updates both internally and to customers in a manner that optimizes updates for performance and compatibility. Other variations upon and modifications to the described embodiments are provided for by the present invention, the scope of which is defined by the following claims.

Detail Description Table CWU:
3 Output Option Flag Description -l RCL Patch has been recalled. -r REL Patch has been released in a Support Plus or Extension Software bundle. -e DEP, Patch has patch dependencies and/or other ODEP dependencies. -c CRIT Patch is flagged has containing critical defect fixes. -a REP Patch was reposted.

CLAIMS:

1. A software-update management system comprising: an input for receiving a request for an update chronology for a target update; an interface for retrieving succession data from a set of databases, said databases having a record for each update, each record indicating all updates that the associated update directly supercedes; and an update chronology generator for generating an update chronology for said target update by accessing the records for said target update and each update indicated by the record for said target update as being directly superceded by said target update, said generator being coupled to said input and to said interface.

First Hit

Generate Collection | Print

L3: Entry 4 of 25 File: PGPB Jan 1, 2004

DOCUMENT-IDENTIFIER: US 20040003266 A1
TITLE: Non-invasive automatic offsite patch fingerprinting and updating system and method

Summary of Invention Paragraph:
[0003] The present invention relates to systems and methods which update existing software across a remote network. The invention relates more specifically to checking for the need for updating and then updating the software as required across a client-server system without the need for human oversight, and without requiring that a target network administrative machine keep copies of software patches.

Summary of Invention Paragraph:
[0006] Each time software is "debugged", a change to that piece of software is created--this change sometimes results in an additional piece of software called a "patch" or "fix". The industry's software vendors often call these patches by the more formal names "Service Packs" or "Support Packs."

Summary of Invention Paragraph:
[0008] Microsoft, for instance, for its Windows NT family of operating system software products has no less than six major "Service Packs" available to be applied to solve problems its customers may experience. More generally, the total number of patches, fixes, solutions, and/or service packs for any given operating system may be enormous.

Summary of Invention Paragraph:
[0009] When an application is installed, it may contain one or more of these operating systems file patches along with the standard computer files. The patches are generally included because the application vendor discovered some anomalous behavior in one or more of the operating system files, and so sent a "fix" in the form of a different version of one of these troublesome files. This would cause relatively little difficulty if only one application vendor performed this service, or if the file modified by the application vendor is used only by that vendor's application. However, this is often not the case.

Summary of Invention Paragraph:
[0014] If the three thousand clients were all running the same workstation operating system, that could mean another twenty-one thousand visits to apply the patches. Remember this all has to be accomplished while installing and patching the internally developed applications and the shrink-wrapped products. Distribution of software patches and files and their subsequent application becomes the first indication of what might be called "administrator agony".

Summary of Invention Paragraph:
[0018] Furthermore, it is not always obvious exactly what patches, if any, a given piece of software has received. Updates don't always clearly announce their presence. So, it is not always clear whether a specific computer has previously received a specific patch. Accordingly, there is a need for improved tools and techniques for updating computers across a network. Such tools and techniques are described and claimed herein.

Summary of Invention Paragraph:
[0020] In various embodiments, the invention facilitates software deployment, software installation, software updating, and file distribution based on software and patch finger printing across multiple operating systems and devices, across a network. Any computer with a network connection and with an update agent running on it may connect to an update server, and then process whatever tasks the administrator has designated for that agent.

Summary of Invention Paragraph:
[0022] Patch fingerprints 902 give a recipe to allow a repository component to determine if a given software package (associated with the patch fingerprint), patch, driver, etc. should be loaded onto a computer in the system. These fingerprints are stored in a patch component database location 900 that may be inside or outside the firewall 214. It may be at a separate location or it may be installed on the update server 528. The repository component also includes an inventory library database 918 that contains basic hardware and software information about each of the network target computers 202, 208. Using the information in the patch fingerprint, the inventory library, and specific information gleaned from each network target computer, the system is able to intelligently recommend which patches and drivers are required for a given computer.

Summary of Invention Paragraph:
[0023] As shown in FIG. 5, the preferred embodiment of the invention employs an additional agent known as the discovery agent 548 installed on the target computer 500, which routinely discovers the hardware and software on that machine. This inventory information is then reported back to an inventory library 918 located somewhere else in the repository component. In addition to the computer inventory, the discovery agents also return scan results for patch fingerprints, which indicate whether it is appropriate to install a specific patch associated with each patch fingerprint.

Summary of Invention Paragraph:
[0024] The Inventory Database thus collects a complete inventory of the software, hardware and current patch fingerprints that are installed on any particular target computer within the network. With this information, the update server 528 can present the user with detailed reports of the current patch status for all computers within the network. This illustrates the number of computers needing the patch as well as the computers already installed with the patch.

Summary of Invention Paragraph:
[0025] In addition, Finger Print definitions 906 are also normally associated with an update package suitable for deployment by the system. Once the need for a particular patch has been established by scanning for its signature(s) on all or any computers within the network it can then be quickly deployed by the administrator by merely selecting the date and time.

Summary of Invention Paragraph:
[0026] In some embodiments, fingerprint definitions 906 may be combined with one or more of the following to form a portable patch definition file: vendor bulletin(s) discussing the patch(es), report(s) prepared by embodiments of the invention for administrators, target computer 500 signature(s), deployment package(s). This patch definition file provides information that can be used to update other networks. The patch definition file (a.k.a. "patch metafile") provides a portable uniform data representation which can be employed by embodiments of the invention to move or replicate patches among update servers 528 of different networks. Suitable networks 100 include without limitation networks that are not connected to the Internet and/or to each other, such as military networks that are isolated to provide greater security. This movement/replication can be done by email, tape write/read,

and/or other conventional data transfer means. The patch metafile may also aid the interchange and interoperability of patches between inventive embodiments supplied by different vendors.

Summary of Invention Paragraph:
[0027] The patches that need to be loaded onto specific target computers are listed on the update server 220 in update lists 222 associated with update agents 204, 210; in the illustration, list 224 is associated with Target1 202, and list 226 is associated with Target2 208. The update lists specify at least one location (through means such as a universal resource locator, or URL) where the patch can be found, and optionally include a date which is the earliest date that the software can be installed.

Summary of Invention Paragraph:
[0028] In operation, the update agent 204 of Target1 202 checks its update list 224 at the onsite or offsite update server 220 to see if a new package should be installed. If one is there, the update agent 204 checks to see if the package is already in memory on the update server 220. If so, the update agent 204 attempts to install the software patch directly from the update server 220. If not, the update agent 204 attempts to install the software patch directly from the package computer location 232. In some instances, this is successful, in which case the update list 224 is updated.

Summary of Invention Paragraph:
[0029] In other cases, a download 218 will be obstructed by the firewall 214. If this happens, the update agent 210 informs the update server 220 and then the update server 220 itself will attempt to retrieve the package and place it in memory 228. From that memory on the update server, the software is installed directly to the target machine.

Summary of Invention Paragraph:
[0030] A monitor checks to see that the software installs properly on the target 202, 208, and then continues checking (or can be notified) to ensure that the updated software runs correctly and that the target computer itself doesn't experience any problems in what appear to be unrelated areas. Should the package fail to install properly, or create problems for the software program that was patched, or create other problems on the target computer, the package can be automatically removed and the computer restored to its preinstalled state or another acceptable state in which the update has been removed or disabled, and the target computer is in a workable state. If the package has been installed on more than one computer, they all can be removed. If the error occurs in the middle of a rollout to many computers, the rollout can be halted and the software removed or disabled. The monitor may be located on the update server 220, on a repository site 600, at least partially in the update agent 204, 210, and/or in a combination of such locations.

Summary of Invention Paragraph:
[0032] The update agent 204, 210 can also be used to survey its own target computer, and this information can be stored in a database offsite or at another location. This information can then be used to determine what updates a given target computer needs in order to have the most appropriate configuration. When a new software patch becomes available, the stored information can be used to determine if a particular target computer needs the patch.

Detail Description Paragraph:
[0045] The present invention provides systems, methods, articles, and signals which help update existing software across a remote network. The invention relates more specifically to updating software across a client-server system with little or no need for human oversight, and without requiring copies of the software patches on an administrative machine on the network whose clients are being updated. The

update is automatic, and it can detect errors within a specific update and automatically rollback a faulty update to leave the network in a usable state.

Detail Description Paragraph:
[0059] System administrators often need to change the software on a specific machine. A new piece of software must be installed for the first time, as when a new application is added to a machine. An already-installed piece of software can be updated, as when a new version of an existing piece of software will be installed on a specific machine; this is also referred to as "replacing" the software. A data file of an existing piece of software can be updated without otherwise changing the software configuration, as when tax tables are updated on an accounting program, or when anti-virus software files are updated. If a problem is discovered in an existing piece of software, then a fix or patch can be installed. Any or all of these changes to the state of a specific machine or machines are referred to in this patent as "installation". Similarly, the words "package," "patch," and "update" should be given the broadest possible meaning. For example, package could refer to an entire program including all the necessary files, to one or more data files, to a software patch to an existing file, to a change to a configuration file, to a *.dll file, a driver file for a specific piece of hardware attached to a computer and/or a computer network, and so on. "Update" refers to at least attempting to install a package on a computer.

Detail Description Paragraph:
[0063] Agents in embodiments of the present invention can receive compressed files to conserve network bandwidth. Compression also enhances security, because decompression errors may indicate that a patch has been tampered with.

Detail Description Paragraph:
[0064] An inventive agent can also resume a download when a mobile target bearing the agent is disconnected and then reconnected to the network at another location, unlike patch management tools that lack agents and therefore download entire service packs or files after being interrupted. Tools lacking agents may also generate uncontrolled spikes in bandwidth utilization as patches are deployed, whereas some embodiments of the present invention permit an update server to be controlled by an administrator so that the server uses only a specified amount of bandwidth per agent connection (bandwidth throttling).

Detail Description Paragraph:
[0065] Conventional patch tools that rely on a permanent LAN/domain connection and lack client agents may rely on a remote registry service, which provides registry information to a remote computer. The remote registry service is not available on the Windows 95, Windows 98, or Windows ME platforms. Such a service can be a security risk in organizations whose client computers are on the Internet, because they allow remote computers to read the registry of a client, thereby providing information that can be used to guide infiltration or other attacks on the client's security. Embodiments of the present invention preferably avoid using a remote registry service, due to the security risk.

Detail Description Paragraph:
[0069] The actual update material that is to be installed on the target computer is often stored at a separate location (known as the package computer) apart from the update server and the target computer. The software update itself can be any of a wide variety of software that can be updated across a network, such as an incremental software patch, a new software program never before installed on the target computer, an update to an old program, software scripts, data files, or even an update of the update agent.

Detail Description Paragraph:
[0070] If a known condition is met, during a placing step 300, a task id is placed on an update task list 222. The known condition could be that the patch is not

currently on the computer, that the administrator has given assent, the owner of the target computer 500 has permission from the owner of the package, the fact that no one has specifically denied placing privileges, or some other known or inventive condition. The update task list located on the update server is associated with a specific target computer 500, and specifies at least one download address where the software update can be found. The download address can be in any format that is understandable to the computers. The invention does not depend upon any specific addressing convention. Two common addressing formats in use currently are the "Universal Resource Locator" and "fully qualified domain name" formats. Other formats are PURLs (Persistent Uniform Resource Locators) and URN's (Uniform Resource Names), and other naming schemes may be known in the future. Other information that may be included in the task identifier, such as a date the download will first be attempted. Multiple download addresses, each of which specifies a location where the software update can be found, may all be associated with a specific software update.

Detail Description Paragraph:
[0071] During a starting task step 304, the software update is at least attempted to be uploaded from the package computer 567 to the update server 528. During an optional choose package computer step 306, if more than one download location is placed on task id list 226, the location that the software update will be downloaded from is chosen. The choice can be made by any known or inventive method, such as using the first location on the list, using the location that a test message returns from most rapidly, using the first available machine, and so on.

Detail Description Paragraph:
[0072] Once a location for the update is known, the software download is attempted from the location of the package computer 548 to the memory 530 of the update server 528. If the download is unsuccessful, then in one inventive method another location from the list of possible locations in the task update list is chosen, and the download of the software update is retried. In some implementations, if the download can't be completed for some reason, the update server 528 waits for a time and tries to download from the package computer 567 again. If the download is successful, then the update server 528 attempts to download 312 the software update to the target computer 500.

Detail Description Paragraph:
[0073] Once the download is in cache or other memory in the update computer a second download 312 is attempted to download the software package from the update server to the target computer. In some embodiments of the method, the second download 312 is delayed 310 by some predetermined criterion. This delay may be from the start of the first download, with the delay period based on an estimate of the time needed to download the software update from the package computer to the update server. The second download may also be delayed to a specific time of day when the target computer 500 has less of a chance of being used, such as after a business closes for the day. Other known or inventive delay criteria may also be used.

Detail Description Paragraph:
[0075] If the monitoring step detects a failure 316, then the task that failed is suspended 318. The first download 308 to the update server 528 could fail, as could the second download from the update server 528 to the target computer 500. If there are multiple target computers having the software update installed, the Nth installation could fail, and so on. Determining results preferably goes beyond simply ensuring that the software update appears to have installed properly, and in some embodiments of the invention extends for a time beyond the installation. For example, one embodiment of the monitor will test a patch application by having it installed on only one target computer, assuring that it downloads properly, installs it and then watching it for some period of time until the administrator who sets the time delay gains enough confidence in the patch to allow it to be applied to other target computers. Should the application of this patch cause

abnormal activity, as noticed by undesirable behaviors either in the program whose software was modified or elsewhere in the computer, the rollout can be automatically suspended until the problem is resolved.

Detail Description Paragraph:
[0084] In a detect success step 408 the target computer 500 sends a message 410 to the update server after the download from the update server to the target computer has completed successfully. The monitor can presume success 404 if a specified time period has passed without noticing or being notified of a failure.

Detail Description Paragraph:
[0085] Failure can be detected in other ways 316, 406. For instance the target computer can notify the monitor that a failure has occurred; a user can notify the monitor through the help desk or through a direct link that a failure has occurred; when a target computer does not contact the monitor within a specified time from the beginning of the second download 312 onto the target machine, a human administrator can declare that a failure has occurred; and so on. Notice that even after the monitor has declared the outcome of a download to be a success, later events, such as an indication of failure from the help desk, can cause the monitor to declare the download to be a failure.

Detail Description Paragraph:
[0087] It is not always clear by looking at a specific target computer 500 what software packages and patches have been installed. The invention includes a method to analyze a target computer 500 to ensure that a given patch has not already been installed on the computer 500 before the invention attempts to install that patch. The following discussion includes references to FIGS. 8 and 9 and continuing reference to FIG. 5.

Detail Description Paragraph:
[0088] A patch fingerprint which defines a specific software update is described in greater detail below. The patch fingerprint is located 800 by monitoring a patch component database location 900 for a new patch fingerprint 902. The word "new" here indicates that the patch has not yet been downloaded into the repository component 600, or for some reason needs to be downloaded into the repository component again, even though it has been downloaded previously. There may be one or many patch component locations; those locations may be on a separate computer connected to the system through a network link, on the update server 528, on the target computer 599, on the package computer 567, on a non-networked location such as a CD, a tape, a floppy disk, etc., or some other known or inventive location.

Detail Description Paragraph:
[0089] Once the patch fingerprint 906 is located 800, it is placed 802 into the repository component 600. The usual method of placement is to download 804 the patch fingerprint 906 into the repository component, but in some embodiments the fingerprint 906 will be on the same file system, so the patch fingerprint will be copied without using the network, such as copying between partitions.

Detail Description Paragraph:
[0090] The illustrated patch fingerprint comprises one or more general inventory install dependencies 912 that can be used to take a high-level look to see if a specific patch can be installed on a machine. It also includes a signature block 910 that can be used to request specific information from a target computer 500, and an existence test 908 which can use the signature block information to determine if a specific patch has been loaded on a machine.

Detail Description Paragraph:
[0091] In some versions of the invention, the inventory install dependencies 912 describe at least some of the necessary software and hardware that must be installed on the target computer 500. These dependencies 912 are compared 808 with

information about the target computer 806 previously stored in the inventory library 918. If the install information and the inventory information don't match, then the patch is not installed. In some versions of the invention a message is sent to at least one administrator containing a list of components required (such as necessary hardware and software) for the install.

Detail Description Paragraph:
[0093] Once the signature information 910 has been sent to the repository component 600, an evaluator 914 evaluates at least a portion of the specific install information requested by the signature block using the existence test 908, and in some instances the inventory install information 912, to determine if the patch is absent 822 on the target computer 500.

Detail Description Paragraph:
[0094] As an optional step, once it has been determined if the patch is absent a message is sent 824 to at least one address associated with an administrator. This message may be sent using a variety of methods, including email, pager, fax, voicemail, instant messaging, SNMP notification, and so on.

Detail Description Paragraph:
[0095] Patch Fingerprint

Detail Description Paragraph:
[0096] With continuing reference to FIGS. 5, 8 and 9, one embodiment of the system verifies that a software package can be or should be installed on a given target computer 500 before attempting installation. To do so, a patch fingerprint 906 is used, e.g., by an agent on a client. The patch fingerprint defines how to determine if a given software package/incremental patch has been previously installed. It may also define a minimum hardware/software configuration necessary for the patch installation. These patch fingerprints 906 are stored in a fingerprint library 904. The fingerprint library 904 is located on a repository component 600. This repository component 600 may be located on the update server 528, or may be in a separate location accessible to the update server 528 and the target computer 500. Some versions of the invention also include an inventory library 918 which contain target inventories. Each target inventory 920 contains the hardware and software information about a defined set of target computers 500. This defined set may include as few as one computer or as many as all of the computers in a given network, or some number in between.

Detail Description Paragraph:
[0097] The fingerprint library 904 can be automatically replenished. In some embodiments, at least one, but possibly several, patch component database locations 900 are monitored 800 for new patches 902. In some embodiments of the invention a signal from the locations 900 indicates to the repository component 600 that new patches 902 are available 800. In the preferred implementation the fingerprint library 904 is updated with new patch fingerprints at specific time intervals. After the repository component 600 is aware of the new patch fingerprint, the patch fingerprint is placed into the repository component 802, usually by using a downloader 924 to download the new patch fingerprint. Patch fingerprints may be entered into the repository components in other ways, however. For example, one or more patch fingerprints may be manually installed into the fingerprint library by an administrator.

Detail Description Paragraph:
[0099] The repository component 600 also contains an inventory library 918. A discovery agent 548, which in some embodiments initially resides on the update server 528, is installed from the update server 528 to the target computer 500 using known or inventive methods. This discovery agent 548, described in greater detail below, inventories at least some of target computer 500's software information 606, hardware information 608 including specific software updates and

patches installed, usage information 604, registry information 612, web information 610, configuration information 614, services 618, file information, patch signatures which have been utilized, etc.

Detail Description Paragraph:
[0102] With this information, a report generator 922 can present a user with detailed reports of the current patch status for all computers within the network, illustrating the number of computers needing the patch, the computers already installed with the patch, computers that can't receive the patch until hardware or software is upgraded and so on. In addition, the report generator 922 can provide a partial or complete inventory of the computers attached to the network. In some embodiments the report generator 922 provides graphical presentations of the inventory for analysis by the administrator, both to track location of hardware as well as to ensure software license compliance. However the repository component 600 also uses the inventory library 918 information as well as detected fingerprint information to distribute relevant signatures 910 from the patch fingerprint 906 to the discovery agent 548, thus greatly optimizing the patch discovery process by eliminating unnecessary scanning work at the target computer 500.

Detail Description Paragraph:
[0104] One optional step to decide if a given software program or patch can be installed is by verifying that the necessary hardware, if applicable, is present, and/or the necessary software is present. For example, some programs may require a specific operating system, some programs may require a certain processor. As an example, if an update of Microsoft Word software is to be installed, it is necessary that Microsoft Word software be on the machine. These high-level dependencies are stored, in some versions, in the inventory install block 912 in the patch fingerprint. The information in the inventory install block is generally high level enough that it can be pulled out of the target inventory 920 of the specific target computer 500 stored in the inventory library 918.

Detail Description Paragraph:
[0105] In some implementations of the invention the patch fingerprint 906 also includes installation dependency information 912. This, as explained above, is information about the target computer 500 that can be expected to be found in the inventory library, and so can be checked without querying the target computer 500. This includes software that should be present (such as a specific version of a program, a patch, a data file or a driver) a hardware component that should be present, or specific hardware and/or software that shouldn't be present.

Detail Description Paragraph:
[0106] If the inventory library does not have an up-to-date inventory for the target computer 500, the discovery agent can be used to scan the target computer 500 for inventory information; it does not necessarily need to also scan simultaneously for signature information. In the preferred implementation, the first time that the discovery agent 548 runs on a given target computer it scans only for inventory information and then loads that information into the inventory library 918; it ignores the patch fingerprint information. At other times when the discovery agent 548 runs it may ignore inventory information and may, rather, be used to look up specific signature information 910 to test for the existence of a specific patch. When the signature block information is looked for, values such as registry entries and INI file values may be inspected for existence, or the actual value may be returned to the repository component 600.

Detail Description Paragraph:
[0107] Each Patch fingerprint comprises a signature block 910 and an existence test 908. The patch signature block is a set of information requests, the information itself to be gleaned from a target computer 500 which will then be used to determine if all necessary bug fix and security patches are installed. Examples of patch signature block information include but are not limited to file, hardware,

registry and configuration information, a specific file name or directory name, all or part of a path that a file is expected to be found in, a specific version number of a file, a created date of a file, a specific file version of a file, and a specific registry value.

Detail Description Paragraph:
[0108] In one implementation the fingerprint library 904 is a SQL database. The patch signatures 910 are extracted from the SQL fingerprint library and then sent to all target computers that meet the dependency criteria for operating system and installed software as specified in the inventory install information 912.

Detail Description Paragraph:
[0109] A preferred implementation employs an XML-based request input file. The result file sent back to the update server 528 also employs XML formatting. This result file contains the signature information for the target computer, and may also contain the software and hardware inventory updates. The inventory and signature information sent to the update server can be quite voluminous, and so are compressed and may also be encrypted in the preferred implementation. The following is a sample patch signature that will gather registry information for Microsoft Outlook as well as the EXEs date and time, and information in the registry:

Detail Description Paragraph:
[0110] Once the discovery agent on the target computer has returned its scan results for the signature, the existence test 908 logic is used by the evaluator 914 to infer whether that particular computer actually has the patch or not. This algorithm minimizes the number of tests that must be done by the evaluator: its sole responsibility is to discover information--allowing the data analysis to be done by the repository component 600 itself. Distributing the workload in this fashion provides a better implementation for scanning and analyzing huge numbers of workstations and servers.

Detail Description Paragraph:
[0111] Each existence test is specific to a given patch. A sample existence test might appear as: if registry QQ contains value ZFILEVAL or (if file Z123.bat was changed on date Dec. 12, 2000 at 11:52 pm and file Z is of size ZFILESIZE) then the patch ZPATCH is present. The preferred embodiment of the patch fingerprint library is an SQL database, but other known or inventive databases can be used.

Detail Description Paragraph:
[0112] Note that a patch fingerprint may also contain dependencies to other Finger Print definitions: for example, "MS-023 IIS Vulnerability Fix" patch might hypothetically require the presence of "Microsoft Windows Service Pack 2". This is used to further optimize where the patch signatures are actually sent. These may sometimes be used in the install dependencies info 912 and other times in the signature block 910, depending on circumstances.

Detail Description Paragraph:
[0113] In addition, fingerprint definitions 906 are also normally associated with a software package 554 suitable for deployment by the system. Once the need for a particular patch has been established by scanning its signature(s) on a computer or all computers within the network, it can then be quickly deployed by the administrator by merely selecting the date and time.

Detail Description Paragraph:
[0114] A fingerprint definition 906 may also contain a logical expression that should be evaluated to assess whether the other elements within the patch signature should be evaluated to TRUE (patched) or FALSE (not patched). The expression is a simple logical statement such as (A AND B).vertline.C where A, B, and C refer to other fingerprint definitions within the patch signature.

Detail Description Paragraph:
[0115] In some implementations the downloader 924 regularly checks the patch component database for new patch fingerprints. When a new patch fingerprint is located, it is downloaded into the repository component. The evaluator compares the dependencies needed for the specific patch implementation listed in the install info 912 with each of the target computer 500 specifications listed in the inventory library. Then an update list is created which may identify all of the target computers 500 that need the patch, all of the target computers that don't possess the patch, all of the target computers that can receive the patch, as they have the necessary dependencies, and/or all of the target computers 500 that have already received the patch. This update list may now be used to update the target computers, and/or may be sent to an administrator by a notifier 916.

Detail Description Paragraph:
[0116] In some instances of the invention the patch component database is owned by someone other than the target computer 500 owner. Only if this patch update host has given permission to the target computer 500 owner will the downloader be allowed to download the new patch fingerprints into the repository component. The permission may comprise a purchase agreement, a lease agreement, subscription for download permission and an evaluation agreement.

Detail Description Paragraph:
[0117] If any modifications are made that may be of interest to the administrator, the notifier 916 will send a notification message containing the new patch updates that have become available or the patch-related state changes that have occurred in his network configuration. Notifications can be sent via e-mail, pager, telephony, SNMP broadcast or Instant Message.

Detail Description Paragraph:
[0123] The update agent of target computer 500 contacts the update server 528 to determine if there is work for the agent 508 to do. The update server 528 determines this by analyzing an agent's update list queue 536. This update list 536 contains, at a minimum, a software location reference 538, but can also contain a date 540 that indicates the earliest date that the software package 554 can be installed, and multiple software location references, if the same software package is available from multiple locations. The types of software 554 that can be updated comprise, without restriction, patch files 556 that update a currently installed software application on the target computer, data files 558, script files 562, new application files 564, executable files, 560 driver updates, new software versions and updates to the update agent file itself 566.

Detail Description Paragraph:
[0124] When the update agent discovers an entry on its associated update list 536, with an appropriate date 540, if any, the installer 510 initially checks to see if a copy of the software package already exists in memory 530 on the update server 528. If found, it then downloads the software package directly from the update server. This situation may arise when a previous target computer 500 has requested the software package 554 from the update server 528.

Detail Description Paragraph:
[0125] If the software package is not found, the installer 510 then attempts to download the update directly from the package computer location given in the software location reference 538 to the target computer memory 502 using its network connection. This will be possible if there is no firewall 526, or if the update server can connect to the package computer location 548.

Detail Description Paragraph:
[0129] In one embodiment the update server will make the packages available to the list of agents one at a time. If an agent 508 or an outcome finder 512 reports that the application of the patch failed, or if the patch puts the agent's target

computer 500 in such a state that it can no longer communicate with the update server, then the update server will suspend the rollout automatically on the administrator's behalf. At this point, the administrator, or some other designated person can be notified 516 of the outcome.

Detail Description Paragraph:
[0130] An outcome finder 512 determines if the software package installation was successful and then communicates its finding to the update server 528. If the outcome is unsuccessful, as discussed above, a restorer 514 places the target computer in an acceptable non-updated state. The outcome finder 512 does not necessarily monitor only the actual software installation; rather it can be set up to watch uses of the software that was patched, the entire target computer, and/or computers that are networked to the target computer, for some designated period of time. The outcome finder can also have different levels of success. For instance, the installation itself (file copying) can be considered a low level of success, while the target computer not misbehaving for a period of time thereafter can be considered a higher level of success, with different actions taken according to the success level. Success or failure can then be monitored as described earlier, and installation retried, suspended, etc. as necessary.

Detail Description Paragraph:
[0133] These update lists 536 facilitate the administrator's designation of pre-built packages, or custom built packages, to be delivered or rolled-out to managed workstations clients or servers, which we refer to as target computers 500. When these packages are to be made available, updates are scheduled by the administrator to be performed by the invention; this may automate a previous task requiring the administrator's visit to a client to install a patch or service pack.

Detail Description Paragraph:
[0142] Referring now to FIGS. 6 and 7, the network 200 may include many different sorts of target computers, each with an agent that may be specifically constructed for the specific target platform. For example, a network running Microsoft Windows PCs, Apple Macintosh computers, and UNIX computers, may have three types of agents. This provides a benefit in that the agent is capable of surveying its target computer and reporting this computer information 602 to the update server 528 and/or to a separate repository site 600 for storage. In some instances of the system, a discovery agent 548 is provided which performs the scan, as discussed elsewhere. In other instances the scan is performed by the update agent 508, or a downloaded script file 562. Hardware configurations 608, software configurations 606, information about the usage of various hardware and software components 604, web sites visited, emails sent and received 610, can all be sent to the offsite location 600. Once this information is available at the update server, an administrator can view the entire managed network from one place.

Detail Description Paragraph:
[0146] Packages are composed of modules representing files, e.g., software files or data files, and scripts, which are sequences of actions to take upon files in the package. Alternatively one or more script file(s) may be included within the package content, and executed by the agent in order to install the patch. In some embodiments of the invention, a human administrator receives notice of the availability of new software patches. In other embodiments, the notices are sent directly to the offsite update server 528 which decides when to roll them out. The offsite update server can be configured to store in permanent memory the packages that have already been stored on each target computer. When a new package becomes available, or during the installation of an existing package, existing evidence of the software packages that need to be installed, as well as information about previous installations, is available in some embodiments at the offsite update server 528, and in other instances at the repository site 600.

Detail Description Paragraph:

[0148] Security and Critical Patch Management, Features

Detail Description Paragraph:
[0149] The present invention provides tools and techniques for managing and distributing critical patches that resolve known security vulnerabilities and other stability issues or enhancements, etc. in various operating systems. Suitable operating systems include, without limitation, all Microsoft operating systems (e.g., 95, 98, ME, NT, W2K, XP, .W2K3), UNIX operating systems (e.g., Linux, Solaris, AIX, HP-UX, SCO, etc), and Novell NetWare operating systems. Operating system product names are the marks of their respective owners.

Detail Description Paragraph:
[0150] In the past, in order to manage security or otherwise critical patches, corporations and other computer users have frequently checked vendor web sites, e.g., by reading news reports or textual alerts posted around the world wide web or were sent notifications via email subscription or newsgroup etc, to find out about new patches. Upon learning that a vendor whose software is used by the corporation has released a new patch to fix or enhance application software, driver software, and/or hardware, the corporation's software administrative personnel have generally had to manually download the latest relevant patches, test them for compatibility with the corporation's machines in various layouts and configurations, and then distribute the patch(es) manually or using their traditional software distribution tools.

Detail Description Paragraph:
[0151] By contrast, the present invention can provide notification 824 of critical updates to computers in a proactive manner, whether or not they have Internet access. It can operate proactively by performing patch downloads without requiring an express administrator command to perform each download. It can also assist with distribution and installation of software updates, software packages, and other data to networked desktop, server, mobile, and other computers.

Detail Description Paragraph:
[0152] One embodiment of the present invention includes content replication through an update server 528 that retrieves the latest critical updates from a master archive such as a package computer 567. Retrieval may use 128-bit SSL or other familiar protocols for secure transmission. As new updates are added to the master archive, the updates' metadata are downloaded automatically to the update servers and/or the fingerprint library 904. If metadata indicates a patch is critical, the patch can be downloaded to the update server and cached there for rapid deployment. Each patch has an associated installer 912, prerequisite signature 910, and other fingerprint identification 906.

Detail Description Paragraph:
[0153] In some embodiments information is sent in one direction only, namely, from the master archive to the update server, thereby enhancing security of the master archive. In addition, in some embodiments all transmitted information is encrypted, CRC (cyclic redundancy code) checked, compressed, digitally signed, and downloaded 308 over a 128-bit SSL connection. The SSL connection employs a secure network protocol that validates and confirms the authenticity of the master archive as the patch source. Other secure network protocols may also be used. In other embodiments, some of these elements are omitted, e.g., no CRC check is done and/or no digital signature is used, etc.

Detail Description Paragraph:
[0154] The update server 528 acts as the patch source for client target computers 500. The update server, which contains the replication service and administrative tools for managing updates and software packages, can scan clients 500 and schedule patch deliveries to them using protocols such as HTTP, HTTPS, and XML. In some embodiments, the update server uses Microsoft's Internet Information Services. The

update server can be implemented to automatically cache critical updates it receives from the master archive. In some embodiments the administrator can set a replication schedule, can trigger replication manually, or can have the replication software in the update server replicate and distribute software automatically in response to expected or measured network inactivity.

Detail Description Paragraph:
[0155] In some embodiments, administrators can create software packages 554, which they can then deploy like any other patch. That is, a "patch" in the general sense need not presuppose a previously installed close-related piece of software that is being modified, but may comprise a piece of software new to the target. For example, a package containing Microsoft Office 2000 could be deployed to every desktop. Administrators of custom applications can similarly create packages to rollout custom applications and their patches. Some embodiment administrators may also utilize built-in software distribution features to distribute any software packages to any target computer.

Detail Description Paragraph:
[0156] In some embodiments the update server 528 is configured with software and/or hardware which displays an enterprise report matrix or other summary of the patch status of the machines in a corporation or other enterprise. The report is displayed to a network administrator and/or other personnel charged with maintaining the enterprise's computer functionality. The administrator influences (and in some cases totally controls) which updates or packages from the update server are pushed to the clients 500, by setting policies, defining groups, responding to alerts, and/or taking other steps which are discussed here or already familiar. In some embodiments the administrator has full control over the deployment of patches, including control of reboots and the power to set or modify client agent policies.

Detail Description Paragraph:
[0157] Patches may be tested internally before they are widely deployed through the enterprise, since a given patch may behave differently in different enterprises. PatchLink.com Corporation ("PatchLink"), which provides commercial software and services for patch management, and which is the assignee of this application and its ancestors, continually researches, tests, and approves patches before they are released by PatchLink. For instance, when a hot fix for the Microsoft W2K (Windows 2000) operating system is released by Microsoft, it may then be installed and tested by PatchLink on two hundred or more different W2K configurations, such as standard W2K, W2K with SQL server, W2K with Office, and W2K with Exchange (marks of Microsoft), and so on, in combination with various service packs and other hot fixes, before it is released by PatchLink to a master archive 567.

Detail Description Paragraph:
[0158] In some embodiments, the client agent 508 checks 332 an intranet-hosted update server to determine which updates are needed at the client in question. It reports gathered information, such as the current configuration 700, back to the update server, which creates the report matrix for the administrator. In some embodiments, the administrator specifies and approves patch deployment using a deployment wizard. Administrator-approved updates and packages are downloaded 312 in the background, thereby reducing inconvenience to users of the computers receiving the download, and then auto-installed according to a schedule set by the administrator. Administrator-defined rules can control the behavior of the patch install process.

Detail Description Paragraph:
[0159] One embodiment of the present invention provides a proactive service that enables administrators to have the embodiment automatically download 308, 312 and install 510 software packages and updates, such as critical operating system fixes and security patches.

Detail Description Paragraph:
[0160] A built-in security feature of some embodiments of the invention uses digital security identification. Before installing 520 a downloaded update on a target 500, this feature verifies the digital certificate, CRC check, compression, and encryption on each file or package. On the update server 528, access to administrative pages and other controls is restricted to authorized administrators. In some embodiments, replication (downloading) of updates uses SSL and the embodiment checks the validity of downloads to the update server; if the SSL certificates do not properly identify a recognized source (e.g., PatchLink.com) then the download fails, and the server sends an email alert to the administrator. In some embodiments, all information in all downloads (master archive to update server, update server to target) is encrypted, CRC checked, compressed, digitally signed, and sent over 128-bit SSL connections only. In other embodiments, these elements are amended (e.g., 40-bit encryption) and/or omitted.

Detail Description Paragraph:
[0161] A patch signature 910 feature permits an embodiment to scan the target 500 and determine if the prerequisite(s) for each patch have been met, e.g., by having the agent check for the proper software version and the proper hardware drivers on the target. The patch signature and the patch fingerprinting features may each be used to make a detection report which is viewable in an enterprise report matrix. A workstation inventory feature uses a discovery agent 508 to pinpoint the needed software and hardware drivers for a target computer. The discovery agent may also scan the target for necessary signatures for fingerprints. PatchLink.com has a master archive which now hosts one of the largest automated patch Fingerprinted repositories in the world.

Detail Description Paragraph:
[0162] A background download 312 feature in some embodiments provides a secure background transfer service with built-in bandwidth throttling, so the network administrator can decide how the bandwidth should be utilized during large deployments. Some embodiments provide administrators with a configurable agent 508 policy which permits them to define the agent's communication interval and operating hours. For instance, an administrator may set the policy to roll out patches to production servers only between midnight and 2:00 am. In some cases, agents may have more than one policy active at a given time.

Detail Description Paragraph:
[0164] Using a download resumption feature, an embodiment detects interruption 316 of a download, e.g., by a service outage. If the target 500 is a mobile workstation, the user can then simply disconnect it and reconnect it at a different location that is not out of service. If the update server can be accessed (via TCP/IP, for instance), the embodiment will resume its download 312 from at or near the point in the download at which it was interrupted, instead of starting again from the beginning to retransmit the entire package.

Detail Description Paragraph:
[0165] A mobile-user support feature allows administrators to deploy patches and software updates to target computers 500 which are not connected to the network when the deployment begins. When a mobile target subsequently connects to the network, the embodiment will automatically scan it and perform the necessary operations to bring that target up to date.

Detail Description Paragraph:
[0166] Embodiments feature client agents 508 which communicate with the update server 528 for secure downloads 312. Using agents also permits increased performance and scalability in enterprise-wide embodiments, permitting a single update server to service thousands of clients. The agents can work across firewalls 116, 214, and operate on any computer 500 with a TCP/IP (or other) connection to

the enterprise network.

Detail Description Paragraph:
[0167] Some embodiments feature support for multi-vendor patches 554, which may also be referred to as "comprehensive patch scanning". The update server 528 is not limited to patches from a single vendor, but instead supports inventive management of patches from multiple vendors. For instance, the update server may coordinate with target agents to scan targets 500 for patch-related security vulnerabilities in software from Microsoft, IBM, Adobe, Corel, Symantec, McAfee, Compaq, WinZip, Citrix, Novell, and many others (marks of the respective companies). This provides a more secure network.

Detail Description Paragraph:
[0168] A grouping feature of some embodiments allows administrators to group selected target computers 500 into sets called, e.g., "containers" or "groups". Operations that are applicable to an individual target computer can then also be applied to containers/groups holding a proper subset of the possible target computers, namely, to every target computer 500 (or every suitable target computer in view of patch signatures and fingerprints) belonging to the specified container. This feature facilitates administrator management of deployments, fingerprint reporting, inventory reporting, mandatory patch baseline policy, and/or client agent policies, depending on the embodiment. For instance, each container may have properties that specify its members, its client agent 508 policies, and its mandatory patch baseline policy. Administrators can select individual clients 500, previously-defined client groups, and/or user-defined groups for deployment. In some embodiments computers can be automatically grouped according to the patch(es) they require.

Detail Description Paragraph:
[0170] A mandatory patch baseline policy feature in some embodiments permits an administrator to specify a minimal (baseline) configuration for one or more of a network's computers. The embodiment will proactively patch operating systems and/or applications to the organizational standards defined by the baseline policy. Supporting patch policies in an enterprise allows the administrator of an inventive embodiment to set patch policies for his/her company whereby no machine 500 in the company, for instance, can fall below a minimum patch level. For example, if mandatory patch baseline policy for a W2K group includes Microsoft Office 2000, Adobe Acrobat Reader 5.0, and Service Pack 2, then all computers placed in this group (whether placed initially on group definition, or placed later) will have at least those pieces of software installed on them.

Detail Description Paragraph:
[0171] A baseline for patches may be associated with a set of computers 500 that is defined by a group (e.g., a user-defined group or an administrator-defined group), or with a set of computers 500 that use a particular operating system (e.g., all W2K computers, regardless of user-or-administrator-defined groups), or with a set of computers 500 that use a particular application (e.g., all computers that use Microsoft Office XP), or with some combination thereof. For example, in some embodiments the administrator could set a baseline policy rule stating that if Microsoft Office XP is installed then the system should automatically patch in Office XP Service Release 1.

Detail Description Paragraph:
[0172] When a mandatory patch baseline policy is used, patches 554 that are dropped (removed) from a target 500 by restoring software from a tape backup, mirrored image, or the like, will be automatically reinstalled after the agent 508 determines the new configuration and that configuration is compared 822 (by the client agent and/or the update server) with the baseline required by the policy. Baseline integrity is thus maintained by these embodiments.

Detail Description Paragraph:
[0173] A mandatory patch baseline policy can be used according to the invention to perform automated detection of unwanted software and removal of that unwanted software from target computers within a network. The mandatory deployment patch to be applied when unwanted software is detected would be to UNINSTALL the unwanted items. For example, one such patch would be "Uninstall KaZaA" which would detect and remove the KaZaA file sharing application from a corporate network, thereby reducing the risk that corporate employees violate copyright laws during the course of the business day, or that they consume all available network bandwidth for entertainment purposes. With government agencies and other large entities, eliminating popup software and other things that distract users from their assigned duties can be a high priority.

Detail Description Paragraph:
[0174] The invention also provides a feature that may be viewed as the logical opposite of mandatory patching to cure vulnerabilities in a network. This logical opposite, which may be termed the "Forbidden Patch" feature, is used to denote a service pack, hotfix or other software that must not ever be installed. Just as the mandatory patch feature is used to auto-fix a vulnerability, the forbidden patch feature is used to prevent the network administrator from installing software that can break an operational configuration. As an example, assume a company has a payroll system that doesn't work with the latest Microsoft Service Pack for Windows2000. If that Service Pack patch is ever deployed manually or automatically to the payroll server(s), the administrator needs to know at once; otherwise nobody gets paid at the end of the week. Some embodiments of the can scan for and detect the presence of "forbidden patches" and alert the administrator. They may also provide rules so that an administrator does not inadvertently deploy a forbidden patch to a machine that should not have that patch installed, regardless of whether the applicable group patching policies say otherwise.

Detail Description Paragraph:
[0175] A patch compliance assurance feature in some embodiments provides administrators with the option of locking a set of patches 554 for a particular computer or a group of computers 500. That is, certain patches are required, but in a manner weaker than in the mandatory baseline feature. If an attempt is made to change target 500 configuration in a way that violates the patch requirement, an email alert message 824 is sent to the administrator. For example, several W2K computers may belong to an administrator-defined group of "IIS Servers" which is subject to patch compliance. For security, the embodiment accordingly locks down all operating system patches and all Internet Information Server patches. If at some later point such patches (including without limitation DLLs) are replaced, then the embodiment will send an email alert to the administrator identifying the computer 500 name and/or the modifications done to it. The newly non-compliant computer(s) and the reason(s) for non-compliance--a summary of discrepancies between their configuration and the locked configuration--can be identified. In some cases, this compliance feature may be used by administrators to identify users who install new software or remove existing software from their machine. Note that this compliance locking feature may be used by some embodiments in conjunction with the mandatory patch baseline feature, to automatically patch a target 500 that is non-compliant. When a locked patch or other software component is removed, it is then automatically reinstalled, and the administrator is notified 824 by email.

Detail Description Paragraph:
[0181] An automatic caching feature in some embodiments causes the update server 528 to automatically download and cache in its local update server storage patches 554 that are marked as critical, high-priority, and/or security-related. The update server notifies the administrator as to which patches are critical and which are cached, and scans for target computers 500 that need the patch. By contrast, non-critical patches may be cached at the update server only after they are first deployed. Caching the critical and security patches before their initial deployment

provides target computers with a readily available source for the patch when the vendor whose software is vulnerable may be overwhelmed by patch requests. During Code Red and Nimda virus attacks, for instance, some users had to wait hours for a connection to the Microsoft web site to get the patches, because of the extremely heavy demand for them. Proactively caching critical and security patches at an inventive update server 528 reduces the risk that operation of target computers 500 will be interrupted or compromised due to a lack of such patches.

Detail Description Paragraph:
[0182] Some embodiments have an intelligent multiple patch deployment feature, which matches patches 554 with operating systems, thereby relieving administrators of the need to expressly and fully identify the operating system used on each target computer. For example, assume Microsoft issued a bulletin for its operating systems which specifies different patches 554 for several different operating system platforms. Administrators using this inventive embodiment need only select "Microsoft operating system" for deployment; they can specify target computers 500 regardless of differences in the operating system details of various specified targets. The embodiment compares 820 patch and operating system requirements for compatibility and for the need for a patch, to ensure that the proper patch gets installed on a given target. Thus, the patch for the Microsoft Windows 98 platform will be installed on a target computer that runs the Windows 98 operating system, the patch for the Microsoft NT platform will be installed on a target computer that runs the NT operating system, and so on. This feature speeds up patch deployment by freeing administrators from the need to manually match patches with targets according to the operating systems (or operating system versions, including prior patches) that are involved.

Detail Description Paragraph:
[0183] Another feature helps detect applicable patches 554 and manage patch interdependencies, thereby helping administrators avoid manually sorting through dozens (or even hundreds) of generally unrelated patches. Instead, the embodiment identifies applicable patches using their metadata, fingerprint, and/or signature data, based on factors such as the operating system involved, the presence (or absence) of other patches, the interdependency of different patches (identifying which patches rely on which other patches to work properly), and the mandatory patch baseline policy (if any). Then the administrator is shown which patches are applicable for the target(s) 500 in question. For example, one embodiment shows IIS patches to administrators only if IIS is installed on a target computer. If used consistently, this feature helps ensure that when a patch is deployed toward a target, that target has the application in question and the patch will install on that target.

Detail Description Paragraph:
[0184] As an example of patch interdependencies, on a Microsoft W2K platform one embodiment will recommend Service Pack 2 to the administrator, and once Service Pack 2 is installed it will then recommend a Security Rollup patch, which depends on Service Pack 2. The embodiment reads both the registry and the file information to correctly perform fingerprinting to validate patch 554 identification.

Detail Description Paragraph:
[0185] Some embodiments allow an administrator to review a history or log of recent operations, and to also uninstall a patch 554 or portion thereof, and rollback effects of deploying the patch to the network. This allows the administrator to undo a patch installation that has caused problems. Lost user data will not necessarily be recovered, but the usual steps taken by a conventional uninstaller can be taken using a restorer 514, such as deleting a DLL, removing a registry entry, restoring a path or other system variable value, and so forth. In addition, the configuration status particular to the embodiment, such as signatures, fingerprints, alerts, and reports, is updated to reflect the problems encountered and/or the removal of the patch. The administrator can also be notified if the

removed patch appears in a patch dependency and/or in the mandatory patch baseline.

Detail Description Paragraph:
[0187] Some embodiments operate according to a selective patch feature, under which patches 554 are not automatically installed unless they are required to meet the mandatory patch baseline policy. In some, patches marked as critical and/or security patches are also installed automatically. In such embodiments, other patches are not installed until they administrator selects them and expressly authorizes their installation; this permits administrators to test patches internally within their organization before installing them on the organization's computers. Once the patch is adequately tested, it can be added to the mandatory patch baseline for the group of targets 500 in question, so that it will be automatically installed when needed.

Detail Description Paragraph:
[0188] Some embodiments support a security policy patch 554 that prevents applications from running on a target machine 500. This provides a policy-driven way to hook into the target computer's file system and stop a particular file (or multiple files) from executing. This could be implemented by patches that rename the executable/DLL file(s) in question and substitute in place thereof code that does nothing, or code that displays an error message to the user, and/or code that notifies the administrator by email.

Detail Description Paragraph:
[0189] Operation of inventive embodiments may be further understood by considering the following example scenarios. In one scenario, as new patches 554 are released by their respective vendors, an update server 528 downloads the corresponding fingerprints from a master archive 567. The embodiment then checks to see if any target computers 500 meet the profile (need the patch in question) by sending the patch's fingerprint to targets for scanning by agents 508. The administrator is notified of the new patch and its potential impact on the network, and a report matrix informs the administrator which targets need the patch and which do not. The administrator selects one or more individual target computers and/or groups, and authorizes deployment. Deployment proceeds as discussed herein. The administrator may set the time of deployment, and decide whether to reboot after the installation.

Detail Description Paragraph:
[0190] In a managed data center scenario, the center's administrator creates a patch group from each cluster of data servers. The administrator can test critical updates received from a master archive 567, and then deploy tested patches 554 on network targets, either all at once, or in stages to groups. Agent policies can help the administrator specify the hours of operation for each group.

Detail Description Paragraph:
[0191] In an embodiment update scenario, the software used by the embodiment is updated by using the embodiment. That is, when a vendor (such as PatchLink.com) provides patches 554 to the software for target agents 508, update servers 528, and/or other embodiment software, those patches can be deployed as discussed herein, using the inventive tools and techniques that would more often be used to deploy patches to operating systems or user applications. For instance, an administrator can select a PatchLink HotFix client patch and deploy it to update client agent software. Client agents may be initially deployed by pushing them to all target computers.

Detail Description Paragraph:
[0194] The invention provides systems, methods, and configured storage media for assuring that software updates are needed, and that the computers have the necessary software and hardware components, then updating the software across a

network with little or no need for human oversight, without requiring copies of the software patches on an administrative machine on the network whose clients are being updated, and which removes the updates from the affected machines, leaving them in a usable state when a problem is discovered during installation or after installation with an installed patch.

CLAIMS:

1. An automated method for updating software in a system having a first target computer in a non-update state connected across a network to an update server in a pre-update state, the system also having a package computer which may be inaccessible to the first target computer and is accessible to the update server, and a repository component accessible to the first target computer and the update server, the method comprising the steps of: putting at least one patch fingerprint which defines a specific software update into the repository component; gathering information about the first target computer; comparing at least a portion of the gathered information with the patch fingerprint to determine if the specific software update is absent from the target computer; placing at least one task identifier on an update task list, the task identifier specifying the first target computer, the task identifier also specifying at least one download address which references a location on the package computer that contains a software update for the first target computer; in response to the task identifier, downloading the software update from the package computer to the update server; and performing a second download of the software update from the update server to the first target computer.

2. The method of claim 1, further comprising the step of providing a patch definition file which is portable and which can be employed to replicate a patch on update servers in a plurality of networks.

3. The method of claim 1, wherein the method operates proactively by performing the download steps without requiring an express administrator command to perform them.

4. The method of claim 1, wherein the method operates proactively by caching a marked patch at the update server before deploying the patch to target computers, the patch marked as at least one of critical, high-priority, and security-related.

5. The method of claim 1, further comprising at least two steps from the following group of security steps: utilizing encryption to secure patch downloads; utilizing cyclic redundancy codes to secure patch downloads; utilizing digital signatures to secure patch downloads; utilizing a secure network protocol such as SSL to secure patch downloads, wherein at least one of the security steps is available in the particular method embodiment.

6. The method of claim 1, wherein the step of downloading the software update from the update server to the first target computer is performed using a background downloading process, thereby reducing inconvenience to a user of the first target computer.

7. The method of claim 1, wherein the step of downloading the software update from the update server to the first target computer is performed using bandwidth-throttled downloading, thereby allowing a network administrator to decide how bandwidth should be employed during a large deployment.

8. The method of claim 1, wherein downloading is performed subject to a policy which limits the hours of operation, and the policy is set by an administrator, thereby allowing the administrator to decide when patch deployments are allowed to occur.

9. The method of claim 1, further comprising preventing downloads of software

updates from the update server to the package computer, thereby enhancing security of the package computer.

10. The method of claim 1, wherein the method further comprises use of a chained installation feature permitting an administrator to have downloaded patches installed on the target computer with fewer reboots than would otherwise be required.

11. The method of claim 1, wherein the method further comprises use of a download resumption feature which detects interruption of a downloading step and then after a reconnection resumes the downloading step from at or near the point in that downloading step at which the interruption occurred, thereby avoiding repetition of the entire downloading step to achieve the download.

12. The method of claim 1, wherein the method further comprises use of a mobile-user support feature which allows an administrator to deploy a patch to the first target computer even though the first target computer is not connected to the network when the task identifier placing step occurs.

13. The method of claim 1, wherein the method comprises downloading multiple patches which originated from multiple vendors.

20. The method of claim 1, wherein the method further comprises use of a mandatory patch baseline policy which specifies at least in part software that should be installed on the first target computer, and the method proactively downloads and installs on the first target computer a patch that is specified in the mandatory patch baseline policy.

21. The method of claim 20, wherein the mandatory patch baseline policy sets a baseline for target computers that use a particular application.

22. The method of claim 20, wherein the mandatory patch baseline policy mandates removal of unwanted software from a target computer.

23. The method of claim 1, wherein the method further comprises use of a forbidden patch feature which specifies software that should not be installed on the first target computer, and the method attempts to prevent such installation from occurring.

24. The method of claim 20, wherein the method further comprises automatically reinstalling a patch that is specified in the mandatory patch baseline policy after software in the patch was dropped from a target computer that is subject to the mandatory patch baseline policy.

25. The method of claim 1, wherein the method further comprises the steps of grouping a proper subset of target computers to form a group, and using a mandatory patch baseline policy to specify at least in part software that should be installed on the target computers in the group.

26. The method of claim 1, wherein the method further comprises use of a patch compliance assurance feature which specifies software that is locked on the first target computer, and the method proactively notifies an administrator if locked software is removed from the first target computer.

28. The method of claim 1, wherein at least the step of downloading the software update from the update server to the first target computer recurs, thereby repeatedly updating a particular file on at least the first target computer.

30. The method of claim 1, further comprising the steps of maintaining a record of recent operations, and rolling back deployment of a patch, thereby allowing an

administrator to undo a target computer patch installation that has caused problems.

31. The method of claim 1, wherein the method further comprises use of a intelligent multiple patch deployment feature which matches patches with target computer operating systems, thereby relieving an administrator of the need to expressly and fully identify the operating system used on the target computer.

32. The method of claim 1, wherein the method installs a security patch on the first target computer, thereby providing an administrator with a policy-driven way to hook into the target computer's file system and stop at least one particular file from executing on the target computer.

33. A configured program storage medium having a configuration that represents data and instructions which will cause at least a portion of a computer system to perform method steps of an automated method for updating software in the system, the system having a first target computer in a non-update state connected across a network to an update server in a pre-update state, the system also having a package computer which may be inaccessible to the first target computer and is accessible to the update server, and a repository component accessible to the first target computer and the update server, the method comprising the steps of: putting at least one patch fingerprint which defines a specific software update into the repository component; gathering information about the first target computer; comparing at least a portion of the gathered information with the patch fingerprint to determine if the specific software update is absent from the target computer; placing at least one task identifier on an update task list, the task identifier specifying the first target computer, the task identifier also specifying at least one download address which references a location on the package computer that contains a software update for the first target computer; in response to the task identifier, downloading the software update from the package computer to the update server; and performing a second download of the software update from the update server to the first target computer.

34. The configured storage medium of claim 33, wherein the method further comprises the step of providing a patch definition file which is portable and which can be employed to replicate a patch on update servers in a plurality of networks.

35. The configured storage medium of claim 33, wherein the method operates proactively by performing the download steps without requiring an express administrator command to perform them.

36. The configured storage medium of claim 33, wherein the method operates proactively by caching a marked patch at the update server before deploying the patch to target computers, the patch marked as at least one of critical, high-priority, and security-related.

37. The configured storage medium of claim 33, wherein the method further comprises at least two steps from the following group of security steps: utilizing encryption to secure patch downloads; utilizing cyclic redundancy codes to secure patch downloads; utilizing digital signatures to secure patch downloads; utilizing a secure network protocol such as SSL to secure patch downloads, wherein at least one of the security steps is available in the particular method embodiment.

38. The configured storage medium of claim 33, wherein the step of downloading the software update from the update server to the first target computer is performed using a background downloading process, thereby reducing inconvenience to a user of the first target computer.

39. The configured storage medium of claim 33, wherein the step of downloading the software update from the update server to the first target computer is performed

using bandwidth-throttled downloading, thereby allowing a network administrator to decide how bandwidth should be employed during a large deployment.

40. The configured storage medium of claim 33, wherein downloading is performed subject to a policy which limits the hours of operation, and the policy is set by an administrator, thereby allowing the administrator to decide when patch deployments are allowed to occur.

41. The configured storage medium of claim 33, wherein the method further comprises preventing downloads of software updates from the update server to the package computer, thereby enhancing security of the package computer.

42. The configured storage medium of claim 33, wherein the method further comprises use of a chained installation feature permitting an administrator to have downloaded patches installed on the target computer with fewer reboots than would otherwise be required.

43. The configured storage medium of claim 33, wherein the method further comprises use of a download resumption feature which detects interruption of a downloading step and then after a reconnection resumes the downloading step from at or near the point in that downloading step at which the interruption occurred, thereby avoiding repetition of the entire downloading step to achieve the download.

44. The configured storage medium of claim 33, wherein the method further comprises use of a mobile-user support feature which allows an administrator to deploy a patch to the first target computer even though the first target computer is not connected to the network when the task identifier placing step occurs.

45. The configured storage medium of claim 33, wherein the method comprises downloading multiple patches which originated from multiple vendors.

52. The configured storage medium of claim 33, wherein the method further comprises use of a mandatory patch baseline policy which specifies at least in part software that should be installed on the first target computer, and the method proactively downloads and installs on the first target computer a patch that is specified in the mandatory patch baseline policy.

53. The configured storage medium of claim 52, wherein the mandatory patch baseline policy sets a baseline for target computers that use a particular application.

54. The configured storage medium of claim 52, wherein the method further comprises automatically reinstalling a patch that is specified in the mandatory patch baseline policy after software in the patch was dropped from a target computer that is subject to the mandatory patch baseline policy.

55. The configured storage medium of claim 33, wherein the method further comprises the steps of grouping a proper subset of target computers to form a group, and using a mandatory patch baseline policy to specify at least in part software that should be installed on the target computers in the group.

56. The configured storage medium of claim 33, wherein the method further comprises use of a patch compliance assurance feature which specifies software that is locked on the first target computer, and the method proactively notifies an administrator if locked software is removed from the first target computer.

58. The configured storage medium of claim 33, wherein at least the step of downloading the software update from the update server to the first target computer recurs, thereby repeatedly updating a particular file on at least the first target computer.

60. The configured storage medium of claim 33, wherein the method further comprises the steps of maintaining a record of recent operations, and rolling back deployment of a patch, thereby allowing an administrator to undo a target computer patch installation that has caused problems.

61. The configured storage medium of claim 33, wherein the method further comprises use of a intelligent multiple patch deployment feature which matches patches with target computer operating systems, thereby relieving an administrator of the need to expressly and fully identify the operating system used on the target computer.

62. The configured storage medium of claim 33, wherein the method installs a security patch on the first target computer, thereby providing an administrator with a policy-driven way to hook into the target computer's file system and stop at least one particular file from executing on the target computer.

First Hit

☐ Generate Collection Print

L3: Entry 10 of 25 File: PGPB Nov 21, 2002

DOCUMENT-IDENTIFIER: US 20020174422 A1
TITLE: Software distribution system

Abstract Paragraph:
A system to securely install software upgrades and patches includes determining which software upgrades and patches should be applied, collecting the software upgrades and patches from vendors by downloading them from the vendor's ftp sites, and installing the software upgrades and patches. Embodiments of the system include interpreting how much memory and disk space is needed to install the software upgrades; interpreting the operating system type, version and architecture the software upgrades apply to; interpreting dependencies on other layered products; interpreting dependencies on other upgrades or patches, interpreting which files and directories will be affected by the installation of the software upgrades; backing out software upgrades and patches.

Summary of Invention Paragraph:
[0011] The abstract of U.S. Pat. No. 5,581,749 titled, System and Method for Maintaining Codes Among Distributed Databases Using a Global Database, patented Dec. 3, 1996, by K. Omar Hossain and James J. Whyte, assigned to The Dow Chemical Company, provides the following description: "A global code system maintains reference records for multiple transaction processing systems on a central database (called a global code database). A client (i.e., a user or an external application) cannot directly create reference records on a transaction processing system. Instead, the client should request the global codes system to create a reference record. The global code system responds, in real time, by adding the record to the global codes database and by distributing the record to one or more transaction processing systems in real time. Similarly, the client cannot directly update or delete reference records on the transaction processing system. Instead, the client should request the global code system to update or delete the reference record. The global code system responds, in real time, by updating or deleting the reference record on the global codes database and by distributing the update or deletion to the transaction processing systems which had been instructed to create the record. In addition to distributing record creations, updates and deletions to the transaction processing systems, the global code system distributes these operations to one or more shadow code systems. Each shadow code system provides a copy of a subset of the reference records for read-only access by remote application programs."

Summary of Invention Paragraph:
[0012] The abstract of U.S. Pat. No. 5,919,247 titled, Method for the Distribution of Code and Data Updates, patented Jul. 6, 1999, by Arthur Van Hoff, Jonathan Payne, and Sami Shaio, assigned to Marimba, Inc., provides the following description: "A system and method for distributing software applications and data to many thousands of clients over a network. The applications are called "channels", the server is called the "transmitter", and the client is called the "tuner". The use of channels is based on subscription. The end-user needs to subscribe to channel before it can be executed. When the end-user subscribes to a channel the associated code and data is downloaded to the local hard disk, and once downloaded the channel can be executed many times without requiring further network access. Channels can be updated automatically at regular intervals by the tuner,

and as a result the end-user is no longer required to manually install software updates, instead these software and data updates are automatically downloaded and installed in the background. This method of automatic downloading of updates achieves for the client the same result as the broadcast distribution of software over a connection based network, but wherein the client initiates each update request without requiring any special broadcast networking infra structure."

Summary of Invention Paragraph:
[0013] The abstract of U.S. Pat. No. 6,151,643 for a automatic updating of diverse software products on multiple client computer systems by downloading scanning application to client computer and generating software list on client computer by Chheng et al, patented Nov. 21, 2000, assigned to Networks Associates, Inc., provides the following description, "A system and method update client computers of various end users with software updates for software products installed on the client computers, the software products manufactured by diverse, unrelated software vendors. The system includes a service provider computer system, a number of client computers and software vendor computer systems communicating on a common network. The service provider computer system stores in an update database information about the software updates of the diverse software vendors, identifying the software products for which software updates are available, their location on the network at the various software vendor computer systems, information for identifying in the client computers the software products stored thereon, and information for determining for such products, which have software updates available. Users of the client computers connect to the service provider computer and obtain a current version of portions of the database. The client computer determines that software products stored thereon, and using this information, determines from the database, which products have updates available, based on product name and release information for the installed products. The user selects updates for installation. The selected updates are downloaded from the software vendor computer systems and installed on the client computer. Payment for the software update and the service is mediated by the service provider computer. Authentication of the user ensures only registered users obtain software updates. Authentication of the software updates ensures that the software updates are virus free and uncorrupted. Changes to the client computer during installation are monitored and archived, allowing the updates to be subsequently removed by the user."

Summary of Invention Paragraph:
[0017] The abstract of a paper titled, Secure Software Distribution System, by Marcey L. Kelley, Lauri A. Dobbs, and Tony Bartoletti, presented at the National Information Systems Security Conference, in Baltimore, Md. Oct. 6 and 7, 1997 includes the following description: "Authenticating and upgrading system software plays a critical role in information security, yet practical tools for assessing and installing software are lacking in today's marketplace. The Secure Software Distribution System (SafePatch) will provide automated analysis, notification, distribution, and installation of security patches and related software to network-based computer systems in a vendor-independent fashion. SafePatch will assist with the authentication of software by comparing the system's objects with the patch's objects. SafePatch will monitor vendor's patch sites to determine when new patches are released and will upgrade system software on target systems automatically."

Summary of Invention Paragraph:
[0018] The abstract of a paper titled, "SafePatch" by Marcey Kelley and Scott D. Elko Oct. 1, 2000 provides the following information provides automated analysis, distribution and notification of security and Y2K patches on network-based computer systems. The software determines what patches need to be installed. It detects patch deficiencies and distributes needed patches as well as the appropriate installation script to Sun systems that run Solaris 2.5.1 or newer.

Summary of Invention Paragraph:
[0019] A secure software distribution system (SSDS) user manual version 0.5 was

prepared May 1998 by Lauri A. Dobbs and Marcey L. Kelley. This manual was available to internal users of the secure software distribution system. SafePatch provides automated analysis, distribution, notification, and installation of security and Y2K patches on network-based computer systems.

Summary of Invention Paragraph:
[0024] The present invention provides a computer-implemented system for secure patch software distribution of software from vendors to client's systems. There is an initial determination of which of vendor's patches have been applied to client's systems. Vendor's patches are then installed on to client's systems. An embodiment of the present invention includes a system that determines which patches should be or should have been applied to a system. Another embodiment of the present invention includes a system for backing out undesirable patches from client's systems. Another embodiment of the present invention includes a system for collecting patches from most vendors by downloading them from the vendor's ftp sites. Another embodiment of the present invention includes a system for interpreting the operating system type. Another embodiment of the present invention includes a system for interpreting the operating system version. Another embodiment of the present invention includes a system for interpreting the operating system architecture the patch applies to. Another embodiment of the present invention includes a system for determining how much memory is needed to install said patch. Another embodiment of the present invention includes a system for determining how dependencies on other layered products affect the installation of a patch. Another embodiment of the present invention includes a system for determining if dependencies on other patches affect the installation of a patch. Another embodiment of the present invention includes a system for determining which files are affected by the installation of a patch. Another embodiment of the present invention includes a system for determining which directories are affected by the installation of a patch.

Summary of Invention Paragraph:
[0025] The present invention provides a system of secure installation of software patches from vendor to client's systems. Embodiments of the present invention include various steps such as the following steps. Determining which patches should be applied to the client's systems. Collecting the patches from the vendors by downloading them from the vendor's ftp sites. Determining which patches should be applied to the client's systems. Collecting the patches from the vendors by downloading them from the vendor's ftp sites. Interpreting how much memory and disk space is needed to install the patches. Interpreting the operating system type, version and architecture the patches apply to. Interpreting dependencies on other layered products. Interpreting which files and directories are affected by the installation of the patches. Interpreting which files and directories are affected by the installation of the patches. Backing out patches that should not have been applied to the clients systems. Installing the patches.

Summary of Invention Paragraph:
[0026] Certain embodiments of the system are known as SafePatch secure distribution software system. This system provides automated analysis, distribution, and notification and installation of security patches on network-based computer systems. SafePatch determines what patches need to be installed. For the patches that are installed, SafePatch checks the permissions and ownership of the files referenced in the patch and ensures that the system software is authentic. SafePatch detects patch deficiencies and distributes needed patches as well as the appropriate installation script to client's systems, and optionally installs those patches.

Detail Description Paragraph:
[0034] System software is constantly changing, making it difficult to maintain the integrity of the software. Often the system software is security-flawed in the beginning. Major network-wide assaults, such as the notorious 1988 Internet Worm

attack, as well as a history of less publicized attacks, exploit these known security flaws to gain illicit access to systems. Vendors are often quick to issue security-patched versions of selected system files. Even when the vendor's issue interim patches to fix the flaws, the new releases of the system software may have overlooked the patches and/or introduced new flaws into the release. This is common in large software companies because the teams that create new releases are often different than the teams that create the interim software patches. This multiplicity of security patches and operating system versions significantly complicates the software authentication effort.

Detail Description Paragraph:
[0036] Vendors are aware of these issues and there is a push toward supplying the customers with "self-installing" patches or similar software installation to assist with the maintenance of software. However, these tools are highly vendor specific and vary wildly in their implementation and effectiveness. The tools applicants have encountered suffer from a common security flaw. These tools attempt to keep track of the patches that they have installed by building a "patch database" file. These tools can be easily fooled into reporting erroneous information because they make no attempt to survey the existing system files using secure cryptographic hashes or even ordinary checksums to ascertain what is actually there. For example, assume a patch to fix vulnerability has been installed using such a tool. Subsequently, an intruder replaces the fixed binary with a Trojan or older flawed version. Using the tool to determine what is installed on the system will indicate that the patch is installed because the tool simply consults the "patch database". Solutions that rely on a database are unreliable and unacceptable for determining the level of "trust" in operating system software.

Detail Description Paragraph:
[0037] Additionally, existing tools do not address the problem of maintenance in a heterogeneous network environment. In an environment where a large mix of vendor systems is employed, the routine maintenance of software versions and interim patches is an administrative nightmare. Learning to operate and manage one vendor's set of software or patch management tools is grueling enough, let alone to do this for all the flavors of UNIX and master their operations manuals. Including other popular operating systems such as Windows NT and MacOS complicates the maintenance task even more. The foregoing examples and other examples known in the art explain why installing patches and upgrades don't get done.

Detail Description Paragraph:
[0038] Software management tools are very much needed to support the assessment and authentication of system software on a network as well as installing and upgrading system software. It would be desirable to be able to know exactly which of one's 250 systems are patched up-to-date, which are not, and what patches are needed for each system. Unfortunately, many organizations exist where an administrator of 250 systems could not determine this in a month's time, and certainly not in a manner that involves the actual examination of installed binary files. An embodiment of the present invention enables an administrator to produce this information within hours of the request, from a single console designed to support this type of inquiry and it will do so by actually examining the files present on these systems.

Detail Description Paragraph:
[0039] How much trust does a new network administrator place in the computer systems he/she just inherited? How much trust does an administrator place in a network recently experiencing suspicious activity? A responsible alternative to full network-wide system authentication would be to shut the machines down for a full re-install of their operating systems, along with the latest complement of security patches. This represents weeks of service disruption, and the assurances it gives will tend to dwindle with time. More often than not, this simply doesn't get done. Again, this is why a software management tool supporting software

authentication is needed.

Detail Description Paragraph:
[0040] A software management tool should also support software re-authentication on a regular basis, commensurate in frequency with the value of the resources being maintained on the systems. An embodiment of the present invention provides system administrators with a fast and highly automated method to authenticate system software, determine security patch versions and detect instances of subsequent tampering. In addition, it will provide a convenient and secure means for automating the installation of required security patches and related system software. Information security demands this capability at its foundation.

Detail Description Paragraph:
[0041] A software management tool should be capable of collecting patches; determining which patches should be or have been applied to a system; and installing and possibly backing out patches. Patches can be collected from most vendors by downloading them from the vendor's ftp sites. To collect the newest releases, these ftp sites should be monitored regularly.

Detail Description Paragraph:
[0042] Once the patches are downloaded to the local system, a software management tool should determine which patches should be or have been applied to a system. This is one of the most difficult tasks to automate in a software management tool. Each patch should be interpreted to determine the operating system type, version and architecture the patch applies to; how much memory and disk space is needed to install the patch; dependencies on other layered products, patches and which files and directories are affected by the installation of a patch. To determine which patches are installed on a system, the files on the system should be compared with files contained in each patch. At present this process is accomplished manually by reviewing the README file associated with each patch.

Detail Description Paragraph:
[0043] If the patch is applicable to the system, then the software management tool can install the patch. Installing a patch usually entails following a set of instructions provided with the patch, or executing a script that will install the patch. Sometimes the patch doesn't work as advertised or it interferes with other applications on the system, so the software management tool should also permit patches to be backed-out. Backing-out a patch is similar to installation (i.e., a set of instructions to follow or a script).

Detail Description Paragraph:
[0044] The present invention provides a system for secure software distribution including determining which patches have been applied to a system. Determining which patches should be or should have been applied to a system. Collecting patches from the supported vendors by downloading them from the vendor's ftp sites. Interpreting the operating system type, version and architecture the patch applies to; how much memory and disk space is needed to install the patch; dependencies on other layered products, patches, or upgrades; and which files and directories are affected by the installation of a patch. Installing and possibly backing-out the patches.

Detail Description Paragraph:
[0045] A serious threat to information resources is the inability to determine and maintain a known level of trust in operating system software. This threat can be minimized if systems are properly configured, use the latest software, and have the recommended security patches installed. However, the time and technique required to assess and install recommended security patches on systems is considerable and too often neglected. This situation is further complicated by the fact that each vendor has his or her own patch distribution and installation process. Some vendors are providing tools to assist with the installation process. However, these solutions,

self-installing patches and installation utilities, fail in at least three ways: the target system is not actually examined, their solutions are vendor specific, and they operate on a single host as opposed to a multi-host networked solution.

Detail Description Paragraph:
[0046] Referring now to the drawings, and in particular to FIG. 1, an illustration of the components of an embodiment of the present invention along with their interaction with the vendor's ftp sites and other network-based computer systems is provided. The embodiment provides secure software by a system capable of determining which patches have been applied to a system. Determining which patches should be or should have been applied to a system. Collecting patches from most vendors by downloading them from the vendor's ftp sites, interpreting the operating system type, version and architecture the patch applies to, how much memory and disk space is needed to install the patch, dependencies on other layered products, patches and which files and directories are affected by the installation of a patch. Installing and possibly backing-out patches.

Detail Description Paragraph:
[0048] Vendor 1, reference numeral 11; vendor 2, reference numeral 12; and vendor 3, reference numeral 13 are shown; however, it is to be understood that an number of vendors can be included. The SafePatch Server 14 monitors the vendor sites for the latest patches and stores them in a non-vendor specific format. The SafePatch Server 14 evaluates target systems on a scheduled basis and installs patches as needed. Client 1, reference numeral 11; client 2, reference numeral 12; and client 3, reference numeral 13 are shown; however, it is to be understood that any number of clients can be included. The System (SafePatch) will provide automated analysis, notification, distribution, and installation of security patches and related software to network-based computer systems in a vendor-independent fashion. This system will allow a network administrator (and users) to query, maintain, and upgrade the software integrity of hundreds of individual systems from a central point through largely automated means. The centralized software system will provide the following services for target systems: rapid system software "trust" determination, automated notification of new vendor security patches, automated determination of patch applicability to target systems, automated installation of security patches and critical system software, ability to "back-out" installed patches, restoring a system's previous state, and collection of site-wide software statistics or metrics on patch status.

Detail Description Paragraph:
[0049] The process SafePatch will use to authenticate the software on a system is more reliable and secure than other vendor-specific tools. SafePatch will compare the target system's objects with the objects from the patch to determine what is actually installed and what needs to be installed. This approach ensures accurate reporting of a system's patch status. It also allows SafePatch to identify objects that do not belong to either the original system distribution or to any released patches.

Detail Description Paragraph:
[0050] The SafePatch Server software is a central service residing on one computer interacting with a host of network-based computer systems, similar to network-based backup software. The network-based computer systems serviced by the SafePatch Server are referred to as target systems or targets. The SafePatch Server is responsible for monitoring vendor's ftp sites and collecting newly released patches. The SafePatch administrator can specify which vendor sites to monitor and which patches to collect (e.g., security, recommended, all). For instance only Solaris 2.8 security patches can be collected. These patches are then converted to a non-vendor specific, machine-readable format and stored in a database. The process of converting patches will involve some human interaction until the vendors adopt a standard patch format. Patches stored in this format are referred to as patch specifications. A patch specification contains information such as the

operating system type, version, and architecture as well as the permissions and ownership for each file and directory manipulated by the patch. A cryptographic checksum for each file is also included in the patch specification to be used for file identification during the evaluation process described later. A patch specification is built for every revision of each collected patch and stored in the patch spec database. By maintaining copies of all patches and all patch revisions, SafePatch can determine what is installed on a system and if it is up-to-date. It is important for SafePatch to collect every revision of a patch because the vendor's ftp sites only provide the latest revision of all patches. Ideally, in the future, vendors will adopt a standard patch format or provide an adjunct for all of their patches (new as well as old patches).

Detail Description Paragraph:
[0051] In addition to collecting patches, the SafePatch Server is responsible for evaluating target systems and installing patches on these systems. The system administrators have full control over the scheduling of target evaluations and patch installations. An administrator can request an evaluation immediately or can schedule evaluations on a repeated basis. The SafePatch Server controls the execution of a request by gathering information from the target systems and giving instructions to install and back-out a patch. To evaluate a system, the SafePatch Server asks the SafePatch Client running on the target system what operating system, version, and architecture are running on the target. It then collects all of the patches from the patch spec database pertaining to this system's operating system, version, and architecture. From these patch specifications a list of directories and files manipulated by the patch is formed. The owner, group, permissions, and checksum (for files only) for each file or directory on the list is checked against the owner, group, permissions, and checksums of the respective directory or file on the target system. This check permits the SafePatch Server to determine which patches are actually installed on the target system without relying on the system's local database. From this information, the SafePatch Server can determine which patches need to be installed on the target system in order to bring it up-to-date. The system administrator can choose to have SafePatch install patches immediately after the evaluation or at some later date and time. The system administrator can also choose not to have SafePatch install the patches and instead report on the patches needed. This allows for the system administrators to dictate which actions SafePatch is to perform on a system.

Detail Description Paragraph:
[0055] For an organization with only a few computer systems (e.g., 1 to 50 computers) all of the same type (e.g., Suns) SafePatch would be configured such that the SafePatch Server resides on one computer and the SafePatch Client software would be installed on all of the computers including the computer running the SafePatch Server software. As an example, consider a network of five Sun workstations running Solaris 2.8. One workstation would run the SafePatch Server software. The SafePatch Server would monitor Sun's ftp site for 2.8 patches and collect only these patches. The SafePatch Client software would reside on all five workstations. The SafePatch Server would control the execution of the evaluations and installations based on the schedule determined by the SafePatch administrator or each target's system administrator.

Detail Description Paragraph:
[0057] In a more complex environment with hundreds or thousands of systems running a variety of operating systems with different architectures there may be multiple SafePatch Servers in order to service the large number of systems and networks. In a complex environment, the idea of centralizing the SafePatch services is taken one step further. Here one or two computers would be configured to support the patch collection and storage function of the SafePatch Server. This centralized patch collection service may be manned by one or two people to assist with the conversion of patches to the standard patch format until vendors adopt a standard format.

Detail Description Paragraph:
[0058] In addition to the one or two patch collection services, one system per subnet or organization would be configured to support the evaluation and installation of patches on a subset of the company's computer systems. The number of systems performing the evaluation and installation service would be determined by administrative domains similar to centralized services (e.g., backups, mail). As shown in FIG. 2, these systems would get their patches from the centralized patch collectors. The system shown in FIG. 2 is generally designated by the reference numeral 20. Vendor 21 and vendor 22 are shown; however, it is to be understood that any number of vendors can be included. The centralized patch collection service 23 monitors the vendor sites for the latest patches and stores them in a non-vendor specific format. The evaluation and installation service 24, evaluation and installation service 25, and evaluation and installation service 26 are shown; however, it is to be understood that any number of evaluation and installation services can be included. Client 27, client 28, client 29, client 30, client 31, and client 32, are shown; however, it is to be understood that any number of clients can be included. The SafePatch Client software would be installed on all systems in the network. This configuration distributes the workload and reduces duplication of effort.

Detail Description Paragraph:
[0059] In addition to the one or two patch collection services, one system per subnet or organization would be configured to support the evaluation and installation of patches on a subset of the company's computer systems. The number of systems performing the evaluation and installation service would be determined by administrative domains similar to centralized services (e.g., backups, mail). These systems would get their patches from the centralized patch collectors as shown in FIG. 2. The SafePatch Client software would be installed on all systems in the network. This configuration distributes the workload and reduces duplication of effort.

Detail Description Paragraph:
[0061] SafePatch performs system-level evaluations to ensure all object modules belonging to an operating system are authentic and patched to the latest revision. An authentic object module is one that corresponds to a vendor's object module. The evaluation process will check to ensure the security patches for a corresponding OS type and version are installed on the target system. Only those objects for which the vendor has issued patches will be checked.

Detail Description Paragraph:
[0063] 1. One patch specification definition file will be generated for each vendor-distributed patch. At a minimum, every object contained within a vendor issued patch will be described within at least one patch specification file. This provided SafePatch with the ability to determine whether an object is authentic and up-to-date.

Detail Description Paragraph:
[0064] 2. Patches collected and processed by SafePatch are categorized as security and/or Y2K patches.

Detail Description Paragraph:
[0065] 3. There are 5 types of patches:

Detail Description Paragraph:
[0066] 1. Patches with no object modules. These patches usually require changes in permissions, group, or ownership on files.

Detail Description Paragraph:
[0067] 2. Patches that replace executables.

Detail Description Paragraph:
[0068] 3. Patches that add new files to a system.

Detail Description Paragraph:
[0069] 4. Patches that include packages that are currently not installed on a system.

Detail Description Paragraph:
[0070] 5. Patches that are related to the processor of a system.

Detail Description Paragraph:
[0073] 2. Collect patches and objects to be considered.

Detail Description Paragraph:
[0075] 4. Iterate over the patch specifications chronologically starting from the latest released patch. Categorize the patch status into one of the following.

Detail Description Paragraph:
[0084] 3. From the Patch Specification Database, collect all patches corresponding to the target OS type, version, and architecture.

Detail Description Paragraph:
[0085] 4. Order all patch specifications by release date. Patches with the latest release date should be first.

Detail Description Paragraph:
[0086] 5. Create a list of unique file objects referenced in the collected patch specifications. Uniqueness is determined by object name. Also create a list of unique directory objects referenced in the collected patch specifications.

Detail Description Paragraph:
[0087] 6. Generate a patch zero (P.sub.0) object containing the intersection of the unique lists (defined in step 5) and the object files and directories that originated from the initial OS release. Add this patch zero object to the end of the patch list.

Detail Description Paragraph:
[0089] 8. Starting with the latest patch specification, iterate over all file objects within the specifications and update the following table:

Detail Description Paragraph:
[0091] If checksum in patch specification matches checksum of object on target then

Detail Description Paragraph:
[0092] If checksum in patch specification matches checksum of object on target then:

Detail Description Paragraph:
[0094] 9. Iterate over all patch specification starting with the latest released patch. Tag each one of the file objects in the patch specification with one of the following codes:

Detail Description Paragraph:
[0095] Determidne patch specification status by looking at the tags on each of the file objects referenced by the patch specification. A patch is installed if one or more file object tags are=(installed) and zero or more file object tags are< (obsolete) or if the total number of file object tags in a patch are equal to the number of installed objects and the number of objects with the package not installed, the patch is considered installed.

Detail Description Paragraph:
[0096] A patch is superseded if all object tags are either obsolete or the package is not installed.

Detail Description Paragraph:
[0097] A patch is not tested if any object tags are Not_Set. Otherwise the patch is needed (should be installed). This includes patches with unknown objects.

Detail Description Paragraph:
[0098] 10. Now test the settings on the directories of installed patches. Iterate over each directory in each installed patch specifications starting with the latest released patch. Update the following table:

Detail Description Paragraph:
[0100] 11. SafePatch Recommended Actions: Determine which patches should be installed. Create two lists: needed_list and not_needed_list. The needed_list will contain those_patch specifications that need to be installed on the target system. The patch specifications on this list are those with a status of needed or installed. The not_needed_list will contain those patch specification that do not need to be installed. Below are the different scenarios where a needed patch may not need to be installed.

Detail Description Paragraph:
[0101] a. Patches that are revisions of other more recent patches do not need to be installed. Also some needed patches may not be needed if the union of other needed patches supersedes it (case 4). If ALL objects in a patch on the needed_list are NOT the latest objects (check ps_latest_obj), then the patch is not needed and is added to the not_needed_list.

Detail Description Paragraph:
[0102] b. Needed patches are then checked against the entries within the vendor specific exceptions file.

Detail Description Paragraph:
[0103] 1. Remove patches specified as ignore records from the needed list. This may include a single revision or multiple revisions.

Detail Description Paragraph:
[0104] 2. Reorder the needed_patch list moving the patches required by patches above the patches requiring them in the needed list.

Detail Description Paragraph:
[0105] 3. Remove any patches from the needed list if it is currently installed on the remote machine.

Detail Description Paragraph:
[0107] a. All patches on the needed_list need to be installed.

Detail Description Paragraph:
[0108] b. Report any ad and ownership changes that differ from what the vendor recommends. To determine changes to acl and ownership iterate through patch specifications. If the_patch specification is installed and acl_match field is false then report ace change and the suggested acl. If the patch specification is installed and the owner_match field is false then report ownership change and the suggested owner.

Detail Description Paragraph:
[0109] c. List patches by status: installed, superseded, needed, and not_tested. All references to the_patch specification with the original product release (e.g.,

Solaris 2.5) specification file should be removed from the list.

Detail Description Paragraph:
[0114] For case 1 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). object_c of patch P.sub.3 is installed on the target system. object_b exists but is not authentic (unknown xsum); object_d can't be found.

Detail Description Paragraph:
[0115] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0116] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0118] Case 2: Some patches improperly installed

Detail Description Paragraph:
[0119] For case 2 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.0, and P.sub.2.2 have been installed. object_b of patch P.sub.3 has been installed on the target system.

Detail Description Paragraph:
[0120] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0121] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0124] For case 3 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.1, P.sub.2.2, P.sub.3, and P.sub.4 have been installed. This system is up-to-date.

Detail Description Paragraph:
[0125] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0126] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0128] Case 4: System never been patched

Detail Description Paragraph:
[0129] For case 4 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patch P.sub.0 has been installed. This system is out of date.

Detail Description Paragraph:
[0130] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest

released patch.

Detail Description Paragraph:
[0131] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0134] For case 5 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.1 and P.sub.2.1 have been installed. This system is out of date.

Detail Description Paragraph:
[0135] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0136] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0139] For case 6 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.1, P.sub.3 and P.sub.4 have been installed. This system needs older patches to be installed that will clobber newer installed patches.

Detail Description Paragraph:
[0140] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0141] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0144] For case 7 there are 5patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.2.2, P.sub.3 and P.sub.4 have been installed except object_b in P.sub.3 has a different owner than on the target system.

Detail Description Paragraph:
[0145] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0146] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0149] For case 7 there are 5 patches (P.sub.1, P.sub.2.1, P.sub.2.2, P.sub.3, P.sub.4) plus the original distribution patch (P.sub.0). Patches P.sub.2.2, P.sub.3 and P.sub.4 have been installed except object_c in P.sub.3 has a different group and object_d in P.sub.4 has different permissions than the target system.

Detail Description Paragraph:
[0150] The first step in the evaluation process is to fill out the following table by iterating over each object in each patch specification starting with the latest released patch.

Detail Description Paragraph:
[0151] The next step is to iterate over each patch specification tagging each of the objects.

Detail Description Paragraph:
[0153] Case 9: Not Patched With Wrong Owner

Detail Description Paragraph:
[0155] Case 10: Not Patched With Wrong ACL

Detail Description Paragraph:
[0157] Case 11 is the same as case 1, but a new file has been added to Patch 5.

Detail Description Paragraph:
[0158] The object /usr/bin/du should be UNKNOWN and Patch 5 is Needed.

Detail Description Paragraph:
[0159] NOTE: to setup this test case you need to move patch 5 to the database directory

Detail Description Paragraph:
[0162] The flow chart, FIG. 1, represents logic incorporated into SafePatch to handle Sun patch anomalies that have been identified to date.

Detail Description Paragraph:
[0163] The following represents two test cases in which the algorithm should pass. (Note: No ignore patches were included due to their simplicity.)

Detail Description Paragraph:
[0165] 1. Determining which patches have been applied to a system.

Detail Description Paragraph:
[0166] 2. Determining which patches should be or should have been applied to a system.

Detail Description Paragraph:
[0167] 3. Collecting patches from most vendors by downloading them from the vendor's ftp sites; interpreting the operating system type, version and architecture the patch applies to; how much memory and disk space is needed to install the patch; dependencies on other layered products, patches, or upgrades; and which files and directories are affected by the installation of a patch.

Detail Description Paragraph:
[0168] 4. Installing and possibly backing out patches.

Detail Description Paragraph:
[0169] The present invention provides a computer-implemented method of secure distribution and installation of vendor software patches onto client systems. The method includes determining which vendor's patches have been applied to a system and installing vendor's patches on a system. The method includes determining which patches should be or should have been applied to a system. The method includes backing out undesirable patches from a system. The method includes collecting patches from most vendors by downloading them from the vendor's ftp sites. The method includes interpreting the operating system type. The method includes interpreting the operating system version. The method includes interpreting the operating system architecture the patch applies to. The method includes determining how much memory is needed to install said patch. The method includes determining how dependencies on other layered products affect the installation of a patch. The method includes determining how dependencies on other upgrades or patches affect

the installation of a patch, determining which files are affected by the installation of a patch, and determining which directories are affected by the installation of a patch.

Detail Description Paragraph:
[0170] A prototype version of SafePatch and a draft SafePatch Users Manual were prepared and tested in May 1998. The prototype version was upgraded to SafePatch version 1.1. An upgraded version of SafePatch, version 1.2.1, provides automated analysis, distribution, notification, and installation of security and Y2K patches on network-based computer systems. SafePatch 1.2.1 determines what patches need to be installed. For the patches that are installed, SafePatch 1.2.1 checks the permissions and ownership of the files referenced in the patch and ensures that the system software is authentic. SafePatch is composed of two components: (1) Patch Server and (2) Vendor Server. Currently, SafePatch 1.2.1 detects patch deficiencies, distributes needed patches, and optionally installs the patches on network-based computer systems. SafePatch 1.2.1 is supported on Sun systems running Solaris 2.5.1 or newer and RedHat Linux systems running versions 6.0 or newer.

Detail Description Table CWU:
1 installed All objects referenced in the patch specification are found on the target system and their checksums match those found within the most recent patch revision. superseded A patch specification is superseded when all objects on are found within a more recent patch specifications. not tested One or more objects referenced in the patch specification are missing on the target machine. needed A patch specification is needed when one or more objects within the patch specification need to be installed on the target system. Through deduction, if a patch specification is not installed, superseded, or not tested, then the patch corresponding to this specification needs to be installed on the target machine. not installed The package associated with this object is not installed on the target system. unknown The checksum of an object located on a target machine does not match the checksum within any of the patch specification files.

Detail Description Table CWU:
3 object.is_latest_installed = True object.ps_latest_obj = current patch id object.ps_matching_xsum = current patch id If patch ownership matches the target's ownership, owner_match = True If patch acl match the target's acl, acl_match = True else object.is_latest_installed = False object.ps_latest_obj = current patch id endif Else if object.is_latest_installed = True then do nothing. Else if object.is_latest_installed = False then

Detail Description Table CWU:
4 object.ps_matching_xsum = current patch id If patch ownership matches the target's ownership, owner_match = True If patch acl match the target's acl, acl_match = true

Detail Description Table CWU:
5 Not_Set object.ps_matching_xsum is Not_Set indicating that the object could not be found or read. U object.ps_matching_xsum is blank meaning the object's checksum does not match any patch specification object checksums. The object is unknown. < current patch specification id <object.ps_matching_xsum : an object more recent than the object listed in the current patch specification has been installed on the target. The object in the current patch specification is obsolete. = object.ps_matching_xsum = current patch specification id : the object in the current patch specification matches the object on the target. The object is installed. > current patch specification id > object.ps_matching_xsum : an object older than the object listed in the current patch specification has been installed on the target. The object in the current patch specification is needed. Pkg_not_installed the current patch specification belongs to a package not installed on the target system.

Detail Description Table CWU:
7 object.Dir_latest_obj = current patch id If patch ownership matches the target's ownership, owner_match = true If patch acl match the target's acl, acl_match = true

Detail Description Table CWU:
8 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4 object_a 0 0 0 0 object_b 0 0 0 0 0 object_c 0 0 .0 slashed. object_d 0 0 0 0 = indicates object is part of a patch but not installed on target system. .0 slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
10 Patch Patch/Object object_a object_b object_c Object_d Status P.sub.0 Installed Unknown < Not_Tested Not Tested P.sub.1 Needed Unknown < Not_Tested Not Tested P.sub.2.1 Needed Unknown Needed P.sub.2.2 Needed Unknown Needed P.sub.3 Unknown = Needed P.sub.4 Not_Tested Not Tested

Detail Description Table CWU:
11 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4 object_a 0 0 0 .0 slashed. object_b 0 0 0 .0 slashed. 0 object_c 0 0 .0 slashed. object_d .0 slashed. 0 0 0 = indicates object is part of a patch but not installed on target system. .0 slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
13 Patch/ Patch Object object_a object_b object_c object_d Status P.sub.0 < < < = Installed P.sub.1 < < < > Needed P.sub.2.1 < < Superseded P.sub.2.2 = = Installed P.sub.3 > = Needed P.sub.4 > Needed

Detail Description Table CWU:
14 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4 object_a 0 0 0 .0 slashed. object_b 0 0 0 0 .0 slashed. object_c 0 0 .0 slashed. object_d 0 .0 slashed. .0 slashed. 0 = indicates object is part of a patch but not installed on target system. .0 slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
16 Patch/ Patch Object object_a object_b object_c object_d Status P.sub.0 < < < < Superseded P.sub.1 < < < < Superseded P.sub.2.1 < < Superseded P.sub.2.2 = < Installed P.sub.3 = = Installed P.sub.4 = Installed

Detail Description Table CWU:
17 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4 object_a .0 slashed. 0 0 0 object_b .0 slashed. 0 0 0 0 object_c .0 slashed. 0 0 object_d .0 slashed. 0 0 0 = indicates object is part of a patch but not installed on target system. .0 slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
19 Patch Patch/Object object_a object_b object_c object_d Status P.sub.0 = = = = Installed P.sub.1 > > > > Needed P.sub.2.1 > > Needed P.sub.2.2 > > Needed P.sub.3 > > Needed P.sub.4 > Needed

Detail Description Table CWU:
20 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4
object_a .largecircle. .largecircle. .0 slashed. .largecircle.
object_b .largecircle. .largecircle. .0 slashed. .largecircle. .largecircle.
object_c .largecircle. .0 slashed. .largecircle. object_d .largecircle. .0 slashed. .largecircle. .largecircle. = indicates object is part of a patch but not installed

on target system. .O slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
22 Patch/ Patch Object object_a object_b object_c object_d Status P.sub.0 < < < < Superseded P.sub.1 < < = = Installed P.sub.2.1 = = Installed P.sub.2.2 > > Needed P.sub.3 > > Needed P.sub.4 > Needed

Detail Description Table CWU:
23 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4
object_a .largecircle. .O slashed. .largecircle. .largecircle.
object_b .largecircle. .largecircle. .largecircle. .largecircle. .O slashed.
object_c .largecircle. .largecircle. .O slashed. object_d .largecircle.
.largecircle. .O slashed. .largecircle. = indicates object is part of a patch but not installed on target system. .O slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
25 Patch/ Patch Object object_a object_b object_c object_d Status P.sub.0 < < < < Superseded P.sub.1 = < < < Installed P.sub.2.1 > < Needed P.sub.2.2 > < Needed P.sub.3 = = Installed P.sub.4 = Installed

Detail Description Table CWU:
26 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4
object_a .largecircle. .largecircle. .largecircle. .O slashed.
object_b .largecircle. .largecircle. .largecircle. .largecircle. .O slashed.
object_c .largecircle. .largecircle. .O slashed. object_d .largecircle.
.largecircle. .O slashed. .largecircle. = indicates object is part of a patch but not installed on target system. .O slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
28 Patch/ Object object_a object_b object_c object_d Patch Status P.sub.0 < < < < Superseded P.sub.1 < < < < Superseded P.sub.2.1 < < Superseded P.sub.2.2 = < Installed P.sub.3 = = Installed P.sub.4 = Installed

Detail Description Table CWU:
29 object/patch P.sub.0 P.sub.1 P.sub.2.1 P.sub.2.2 P.sub.3 P.sub.4
object_a .largecircle. .largecircle. .largecircle. .O slashed.
object_b .largecircle. .largecircle. .largecircle. .largecircle. .O slashed.
object_c .largecircle. .largecircle. .O slashed. object_d .largecircle.
.largecircle. .O slashed. .largecircle. = indicates object is part of a patch but not installed on target system. .O slashed. = indicates object is part of a patch and is installed on target system. = indicates object is not part of the patch.

Detail Description Table CWU:
31 Patch/ Object object_a object_b object_c object_d Patch Status P.sub.0 < < < < Superseded P.sub.1 < < < < Superseded P.sub.2.1 < < Superseded P.sub.2.2 = < Installed P.sub.3 = = Installed P.sub.4 = Installed

CLAIMS:

1. A computer-implemented method of secure installation of vendor's software patches on client's systems, comprising the steps of: determining which of said software patches should be applied to said client's systems collecting said software patches from said vendors by downloading them from said vendor's ftp sites distributing said software patches to said client's systems installing said software patches

2. The computer-implemented method of secure installation of vendor's software

patches on client's systems of claim 1, include steps of: determining which software patches should be applied to said client's systems collecting said software patches from said vendors by downloading them from said vendor's ftp sites distributing said software patches to said client's systems Interpreting which files will be affected by the installation of said software patches interpreting which directories will be affected by the installation of said software patches installing said software patches

3. The computer-implemented method of secure installation of vendor's software patches on client's systems of claim 1, include steps of: determining which software patches should be applied to said client's systems collecting said software patches from said vendors by downloading them from said vendor's ftp site distributing said software patches to said client's systems interpreting which files and directories will be affected by the installation of said software patches interpreting how much memory and disk space is needed to install software upgrades and installing said software patches

4. The computer-implemented method of secure installation of vendor's software patches on client's systems of claim 1, include steps of: determining which software patches should be applied to said client's systems collecting said software patches from said vendors by downloading them from said vendor's ftp sites distributing said software patches to paid client's systems interpreting the operating system type, version and architecture said software patches apply to interpreting which files and directories will be affected by the installation of said software patches interpreting how much memory and disk space is needed to install said software patches interpreting dependencies on other layered products installing said software patches

5. The computer-implemented method of secure installation of vendor's software patches on client's systems of claim 1, include steps of: determining which software patches should be or should have been applied to said client's systems collecting said software patches from said vendors by downloading them from said vendor's ftp sites distributing said software patches to said client's systems interpreting the operating system type, version and architecture said software patches apply to interpreting which files and directories are affected by the installation of said software patches interpreting how much memory and disk space is needed to install said software patches interpreting dependencies on other layered products installing said software patches backing-out said software patches that have been applied to said client's systems

6. A computer-implemented method of secure distribution of vendor's upgrades and patches to client's systems, comprising: determining which of vendor's upgrades and patches have been applied to client's systems determining which said software upgrades and patches should be or should have been applied to said clients systems

7. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 6, include: collection of said patches and upgrades from said vendor's and downloading said patches and upgrades to client systems

8. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: interpreting the operating system type

9. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: interpreting the operating system version

10. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: interpreting the operating system architecture the patch applies to

11. The computer-implemented method of secure distribution of vendor's upgrades and

patches of claim 7, include the step of: determining how much memory is needed to install said patch and upgrades

12. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: determining how dependencies on other layered products affect the installation of said patches and upgrades

13. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: determining how dependencies on other patches, or software upgrades affect the installation of a patch

14. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: determining how dependencies on other software upgrades affect the installation of a patch

15. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: determining which files will be affected by the installation of a patch

16. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: determining which directories will be affected by the installation of a patch

17. The computer-implemented method of secure distribution of vendor's upgrades and patches of claim 7, include the step of: Checking the permissions and ownership of the files referenced in the patch and ensuring that the system software is authentic